TECHNICAL REPORT TR-200 OCT. 1972
NGL 21-002-008 AND
N00014-67-A-0239-0021
(NR-044-431)

F G R A A L

TECHNICAL DOCUMENTATION

BY

C. K. MESZTENYI
H. BREITENLOHNER
J. C. YEH

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

TECHNICAL REPORT TR-200 OCT. 1972
NGL 21-002-008 AND
N00014-67-A-0239-0021
(NR-044-431)

F G R A A L

TECHNICAL DOCUMENTATION

BY

C. K. MESZTENYI
H. BREITENLOHNER
J. C. YEH

ABSTRACT

THIS REPORT DESCRIBES THE IMPLEMENTATION OF FGRAAL, A FORTRAN EXTENDED GRAPH ALGORITHMIC LANGUAGE (TECHNICAL REPORT TR-179) FOR THE UNIVAC 1108. THE REPORT CONTAINS THE DESCRIPTION OF THE IMPLEMENTED DATA STRUCTURES FOR SETS, LISTS AND GRAPHS. IT SUMMARIZES THE CHANGES MADE FOR THE RALPH COMPILER TO ACCOMODATE THE SPECIAL STATEMENTS OF FGRAAL. IT GIVES THE CALLING SEQUENCES GENERATED BY THE CHANGED COMPILER. IT DESCRIBES THE OBJECT TIME SUBROUTINE PACKAGE.

THIS RESEARCH WAS SUPPORTED IN PART BY THE OFFICE OF NAVAL RESEARCH UNDER GRANT N00014-67-A-0239-0021(NR-044-431), AND THE NATIONAL AERONAUTICS AND SPACE ADMINISTRATION UNDER GRANT NGL 21-002-008.

TABLE OF CONTENTS

1. INTRODUCTION

A GRAPH ALGORITHMIC LANGUAGE HAS BEEN DEFINED AS AN EXTENSION OF ALGOL AND FORTRAN IN THE TECHNICAL REPORTS TR-158 AND TR-179, RESPECTIVELY. FGRAAL, THE FORTRAN VERSION OF THIS LANGUAGE, HAS BEEN IMPLEMENTED FOR THE UNIVAC 1108. THIS REPORT CONTAINS DETAILS OF THIS IMPLEMENTATION. FOR QUESTIONS REGARDING THE DEFINITION OF THE LANGUAGE AND ITS USAGE ONE SHOULD CONSULT THE TECHNICAL REPORT TR-179. THE APPENDIX OF THIS REPORT CONTAINS A TABLE SUMMARIZING THE SPECIAL FEATURES OF THE LANGUAGE.

THE UNIVAC 1108 IS A WORD ADDRESSABLE COMPUTER WITH 36 BIT WORDS. IT HAS PARTIAL WORD INSTRUCTIONS THUS CERTAIN DATA PACKING INTO ONE WORD CAN BE IMPLEMENTED EFFICIENTLY. IT ALSO HAS A SET OF MASKED SEARCH INSTRUCTIONS WHICH ARE USED FOR CERTAIN SET OPERATIONS IN FGRAAL VERY EFFICIENTLY.

IN CHAPTER 2 SOME HINTS ARE GIVEN FOR ACHIEVING EFFICIENCY BOTH IN STORAGE AND EXECUTION TIME, BASED ON THE IMPLEMENTED DATA STRUCTURE.

THE FGRAAL IMPLEMENTATION CAN BEST BE DESCRIBED IN THREE PARTS: THE DATA STRUCTURES, THE COMPILER AND THE OBJECT TIME LIBRARY PACKAGE. THE PHYSICAL REPRESENTATION OF DATA STRUCTURES FOR THE VARIOUS DATA TYPES ARE SHOWN IN CHAPTER 3, WHILE CHAPTER 4 GIVES A FULL EXAMPLE OF A GRAPH AND THE ASSOCIATED DATA STRUCTURE NECESSARY FOR ITS REPRESENTATION. THE FGRAAL COMPILER IS A MODIFICATION OF THE RALPH COMPILER FOR FORTRAN V AND MAD. THE MAJOR CHANGES IN THE COMPILER ARE DESCRIBED IN CHAPTER 5, AND CHAPTER 6 GIVES SPECIAL INSTRUCTION SEQUENCES GENERATED BY THE FGRAAL COMPILER FOR THE VARIOUS FGRAAL STATEMENTS. THE OBJECT TIME LIBRARY PACKAGE HAS BEEN WRITTEN IN UNIVAC ASSEMBLY LANGUAGE TO CUT DOWN THE OVERHEAD. THE ROUTINES IN THE PACKAGE ARE GROUPED, AND THEY ARE DESCRIBED IN CHAPTER 7.

WE TAKE THIS OPPORTUNITY TO THANK PROF. V. R. BASILI FOR HIS HELP AND ADVICE IN PREPARING THIS REPORT.

2. HINTS FOR EFFICIENT PROGRAMMING IN FGRAAL

THE READER SHOULD BE ABLE TO GAIN SUFFICIENT INFORMATION ABOUT MODIFYING HIS PROGRAM TO ACHIEVE MAXIMUM EFFICIENCY UNDER THE GIVEN IMPLEMENTATION BY READING THIS CHAPTER ALONE, HOWEVER TO UNDERSTAND FULLY WHY THESE RECOMENDATIONS FOR EFFICIENCY ARE VALID, THE REST OF THIS REPORT MUST BE READ.

THE PRIMARY GOAL OF FGRAAL IS THE EASY IMPLEMENTATION OF GRAPH ALGORITHMS ARISING IN APPLICATION, USUALLY THE EASY IMPLEMENTATION OF THE ALGORITHM ALSO MEANS THAT THE PROGRAM CAN BE EASILY DEBUGGED AND DOCUMENTED, UNFORTUNATELY AN EASILY IMPLEMENTED ALGORITHM CAN BE VERY INEFFICIENT IN MEMORY SIZE REQUIREMENTS AND IN EXECUTION TIME, OBVIOUSLY, THE EFFICIENCY OF FGRAAL PROGRAMS DEPEND ON THE ARRANGEMENT OF SPECIAL DATA (SETS, PROPERTIES, ETC.) IN THE IMPLEMENTED FGRAAL SYSTEM, THE ARRANGEMENT OF THESE DATA ARE DESCRIBED IN THE NEXT CHAPTER, IN THIS CHAPTER, WE GIVE SOME HINTS TO ACHIEVE EFFICIENT PROGRAMS BASED ON THE IMPLEMENTED DATA STRUCTURE IN FGRAAL.

GENERALLY WE SUGGEST THAT THE PROGRAM BE WRITTEN AND DEBUGGED WITH AN EFFICIENT IMPLEMENTATION IN MIND, THIS DOES NOT MEAN THAT THE FIRST VERSION BE AS EFFICIENT AS POSSIBLE, ONLY THAT IT LEND ITSELF TO EASY IMPROVEMENT. ONCE A PROGRAM IS DEBUGGED, THE EFFICIENCY OF THE PROGRAM CAN BE IMPROVED BY INSERTING FORTRAN EQUIVALENCE AND DEFINE STATEMENTS, LINEAR ARRAYS AND SOME CHANGE OF THE ORIGINAL PROGRAM. THESE TYPES OF IMPROVEMENT WILL BE ILLUSTRATED IN THE FOLLOWING SECTIONS.

2.1. EFFICIENT USE OF SETS

* STORAGE CAN BE SAVED DURING EXECUTION
* OF A PROGRAM IF SETS ARE MADE EMPTY
* AS SOON AS THEY ARE NOT NEEDED.

THIS IS BECAUSE EACH SET WITH NO ELEMENT (EMPTY) OR WITH ONE ELEMENT (ATOMIC) USES ONLY ONE MEMORY LOCATION WHICH WAS ASSIGNED TO IT BY THE COMPILER.

* IT IS ADVANTAGEOUS TO KEEP THE NUMBER
* OF NOT EMPTY OR ATOMIC SETS UNDER 15
* AT ANY TIME DURING EXECUTION.

SETS WITH TWO OR MORE ELEMENTS ARE REPRESENTED IN EITHER COLUMN OR BLOCK FORM (SEE NEXT CHAPTER FOR DETAILS). THE COLUMN FORM REPRESENTATION OF SETS ARE ATTACHED TO THE UNIVERSAL SEQUENCE AND AS SUCH THEY DO NOT REQUIRE EXTRA STORAGE SPACE. FGRAAL PROVIDES UP TO 15 SETS TO BE REPRESENTED IN COLUMN FORM, AS SOON AS MORE

THAN 15 SETS HAVE TWO OR MORE ELEMENTS, SOME OF THE SETS WILL AUTOMATICALLY BE TRANSFORMED INTO BLOCK FORM. A SET WITH N ELEMENTS OCCUPIES 2*N/3 MEMORY LOCATIONS WHEN IT IS IN BLOCK FORM. THE RESULT OF A SET OPERATION IS ALWAYS PLACED IN COLUMN FORM UNLESS IT IS EMPTY OR ATOMIC. SINCE SETS IN BLOCK FORM OCCUPY EXTRA SPACE AND REQUIRE EXTRA TIME TO GET THEM INTO THIS FORM, ONE AVOIDS THIS OVERHEAD BY KEEPING THE NUMBER OF NOT EMPTY OR ATOMIC SETS UNDER 15 AT ANY TIME DURING EXECUTION.

ONE CAN KEEP DOWN THE NUMBER OF SETS BY USING EQUIVALENCE STATEMENTS, E.G. ASSUME THAT THE SETS S AND T ARE USED IN TWO NON-OVERLAPPING SEGMENTS OF A PROGRAM. AFTER DEBUGGING THE PROGRAM, ONE CAN MAKE THE TWO SETS EQUIVALENT BUT STILL KEEP THEIR NAME:

```
SET  S , T
EQUIVALENCE  (S,T)
(SEGMENT 1 USES S)
(SEGMENT 2 USES T)
```

2.2. SET EXPRESSIONS AND ASSIGNMENTS

```
*   THE MODIFICATION OF A SET BY OTHER SETS
*   IS MOST EFFICIENTLY IMPLEMENTED BY A
*   SEQUENCE OF SIMPLE SET ASSIGNMENT OF THE
*   FORM  S = S .OP. X , RATHER THAN BY USE
*   OF A COMPLEX EXPRESSION.
```

THE PRESENT IMPLEMENTATION OF FGRAAL DOES NOT MAKE ANY ATTEMPT TO OPTIMIZE THE EVALUATION OF COMPLEX SET EXPRESSIONS. E.G. THE SET EXPRESSION

```
( S .DF. X ) .UN. Y
```

IS EVALUATED FIRST BY USING A TEMPORARY SET T' FOR

```
S .DF. X  --->  T'
```

AND THE FINAL RESULT IS OBTAINED BY THE EVALUATION OF

```
T' .UN. Y
```

A VERY FREQUENTLY OCCURING SET ASSIGNMENT STATEMENT IS RECOGNIZED BY THE FGRAAL COMPILER. THIS STATEMENT,

```
S = S .OP. X
```

MODIFIES A SET S WITH AN OTHER SET X BY TAKING THE UNION, DIFFERENCE, INTERSECTION OR SYMMETRIC SUM OF THEM, I.E. OP IS ANY

ONE OF UN, DF, IT OR SM. THIS STATEMENT IS IMPLEMENTED SUCH THAT THE RESULT OF THE SET EXPRESSION, S.OP.X, IS GENERATED IN THE SPACE PROVIDED FOR S. THIS FEATURE OF FGRAAL MAKES IT MORE EFFICIENT TO USE MORE STATEMENTS INSTEAD OF ONE STATEMENT WITH COMPLEX EXPRESSION. E.G. THE EXECUTION OF THE TWO STATEMENTS,

```
S = S .DF. X
S = S .UN. Y
```

IS MORE EFFICIENT THEN THE EXECUTION OF THE COMBINED STATEMENT

```
S = ( S .DF. X ) .UN. Y
```

2.3. PROPERTIES

* USING FORTRAN ARRAYS FOR STORING
* PROPERTY VALUES CAN BE MORE EFFICIENT
* THAN USING DECLARED PROPERTIES.

IN THE IMPLEMENTED FGRAAL DATA STRUCTURE, THE PROPERTY VALUES WITH PROPERTY IDENTIFIERS ARE LINKED TO THE ELEMENTS IN THE UNIVERSAL SEQUENCE. THIS REQUIRES TWICE AS MANY MEMORY LOCATIONS AS THE ACTUAL MEMORY SPACE NEEDED TO STORE THE PROPERTY VALUES ONLY. THE RETRIEVAL AND MODIFICATION OF A PROPERTY VALUE OF AN ELEMENT IS ACCOMPLISHED BY A SEARCH IN THE LINKED PROPERTY BLOCKS OF THE ELEMENT. CLEARLY, THIS REQUIRES MUCH MORE TIME THEN THE RETRIEVAL AND MODIFICATION OF PROPERTY VALUES IF THEY COULD BE ARRANGED IN A FORTRAN TYPE OF LINEAR ARRAY. SINCE THE ELEMENTS OF THE UNIVERSAL SEQUENCE ARE IDENTIFIED BY POSITIVE INTEGERS, ATOMIC SETS CAN BE USED AS INDICES FOR DIMENSIONED VARIABLES. THIS FEATURE OF FGRAAL MAKES IT POSSIBLE TO CHANGE PROPERTY VARIABLES INTO FORTRAN TYPE DIMENSIONED VARIABLES TO ACHIEVE BETTER EFFICIENCY.

THERE ARE SOME RESTRICTIONS ON THE USE OF LINEAR ARRAYS FOR PROPERTIES:

(1) THE USE OF LINEAR ARRAYS FOR PROPERTY VALUES DEFINES THE PROPERTY FOR ALL VALID INDEX VALUES CORRESPONDING TO ELEMENTS IN THE UNIVERSAL SEQUENCE. A SPECIAL VALUE MUST BE USED IF ONE WANTS TO MAKE DISTINCTION FOR UNDEFINED PROPERTY FOR AN ELEMENT IN THIS REGION.

(2) THE ELEMENTS FOR WHICH THE PROPERTY IS DEFINED SHOULD OCCUPY A CONTIGUOUS AREA IN THE UNIVERSAL SEQUENCE IF A LINEAR ARRAY IS USED. FURTHERMORE, THE FIRST OF THESE ELEMENTS, AND THE SIZE OF THE REGION SHOULD BE KNOWN.

WITH THESE RESTRICTIONS IN MIND, ONE COULD ATTEMPT TO CHANGE A DEBUGGED FGRAAL PROGRAM TO ACHIEVE HIGHER EFFICIENCY. THE STEPS TO

BE TAKEN FOR THIS CHANGE ARE AS FOLLOWS:
ASSUME THAT THE PROPERTY P IS USED FOR ELEMENTS IN A CONTIGUOUS AREA OF THE UNIVERSAL SEQUENCE, THE FIRST OF THESE ELEMENTS IS X1, AND THE SIZE OF THIS AREA IS NX.
THEN REPLACE THE ORIGINAL DECLARATION STATEMENT

```
PROPERTY  P
```

WITH THE FOLLOWING TWO STATEMENTS

```
DIMENSION  PA(NX)
DEFINE  P(X) = PA(X-X1+1)
```

ANY APPEARANCE OF THE STATEMENT,

```
REMOVE P
```

MUST BE REPLACED WITH A SMALL LOOP

```
      DO ZZZ I=1,NX
ZZZ   PA(I) = V
```

WHERE V IS A SPECIALLY DEFINED VALUE FOR 'UNDEFINED' PROPERTY. IN THIS CASE, THE SPECIAL FUNCTION,

```
CHECK(P,X)
```

SHOULD BE REPLACED BY THE EXPRESSION

```
P(X) .NE. V
```

3. FGRAAL STORAGE STRUCTURE ON THE UNIVAC 1108.

THIS CHAPTER DESCRIBES THE REPRESENTATION OF THE SPECIAL DATA STRUCTURES DURING THE EXECUTION OF AN FGRAAL PROGRAM. IN SECTION 1, THE DYNAMIC STORAGE IS DESCRIBED. THIS STORAGE HOLDS ALL THE SPECIAL DATA OF THE FGRAAL PROGRAMS, EXCEPT FOR THE HEADERS CORRESPONDING TO THE DECLARED VARIABLES: SETS, LISTS, PROPERTIES AND GRAPHS. SECTION 2 DESCRIBES THE REPRESENTATION OF THE UNIVERSAL SEQUENCE, AND THE SETS, LISTS, PROPERTIES AND GRAPHS IN THE DYNAMIC STORAGE. SECTION 3 SHOWS THE REPRESENTATION OF A CONTRACTED GRAPH, THAT IS, THE GRAPH REPRESENTATION MODIFIED BY THE LIBRARY ROUTINE CONTR.

3.1. DYNAMIC STORAGE.

THE DYNAMIC STORAGE AREA IS THE WORKING SPACE OF THE FGRAAL SYSTEM AND IT IS A CONTIGUOUS ARRAY OF COMPUTER WORDS WHERE ALL DATA VALUES EXCEPT THOSE ASSIGNED BY THE COMPILER ARE STORED. THE SIZE OF THIS STORAGE AREA VARIES AND CAN BE OPTIONALLY DESIGNATED BY THE USER. LOGICALLY SPEAKING, THE DYNAMIC STORAGE AREA CAN BE SUBDIVIDED INTO THREE DYNAMIC, VARIABLE-SIZED PORTIONS. THE UPPER PORTION IS THE UNIVERSAL SEQUENCE, THE MIDDLE PORTION IS THE UNUSED STORAGE AREA AND THE LOWER PORTION IS THE LINKED-BLOCK AREA. THE TWO END PORTIONS, STARTING FROM THE TWO END BOUNDARIES OF THE DYNAMIC STORAGE AREA EXPAND THEIR SIZES BY SEIZING STORAGE FROM THE MIDDLE PORTION. WHENEVER THESE TWO END-PORTIONS MEET (AND THE MIDDLE PORTION DISAPEARS) THE AVAILABLE STORAGE IN THE DYNAMIC STORAGE AREA IS EXHAUSTED AND PROGRAM EXECUTION IS TERMINATED WITH AN APPROPRIATE MESSAGE.

IN FGRAAL, FROM THE DATA STRUCTURE POINT OF VIEW, THERE ARE TWO TYPES OF STORAGE STRUCTURE : (1) LINEAR ARRAY, (2) LINKED-BLOCK. THE LINEAR ARRAY IS USED MAINLY FOR THE REPRESENTATION OF THE UNIVERSAL SEQUENCE, WHILE THE LINKED-BLOCK STRUCTURE IS USED FOR THE REPRESENTATION OF ALL OTHER TYPES OF DATA (E.G. STAQUE, GRAPH, PROPERTY, ETC.). THE LINEAR ARRAY (OR THE UNIVERSAL SEQUENCE) IS LOCATED ON THE UPPER PORTION OF THE DYNAMIC STORAGE AREA, THE LINKED-BLOCKS ARE LOCATED ON THE LOWER PORTION OF THE DYNAMIC STORAGE AREA. HOWEVER, THE STORAGE MANAGEMENT ROUTINE KEEPS TRACK OF THE STORAGE ALLOCATED TO EACH PORTION AND PREVENTS THEM FROM OVERLAPPING INTO EACH OTHER. THE STORAGE MANAGEMENT ROUTINE ALSO MAINTAINS A SET OF GLOBAL POINTER VARIABLES WHICH ARE UPDATED WHENEVER THE STATUS OF THE DYNAMIC STORAGE CHANGES. THE VALUES OF THESE POINTERS ARE EITHER RELATIVE OR ABSOLUTE ADDRESSES. A RELATIVE ADDRESS, I, REFERS TO THE LOCATION OF D(I), WHERE ARRAY D IS THE DYNAMIC STORAGE AREA. AN ABSOLUTE ADDRESS REFERS TO THE AC-

TUAL LOCATION IN THE COMPUTER MEMORY. THE INITIAL (I.E. BEFORE ANY DATA IS CREATED) VALUES OF THESE POINTERS ARE AS FOLLOWS : (NOTE : LOC(A) DENOTES THE ABSOLUTE LOCATION OF VARIABLE A).

NAME	INITIAL VALUE
NU - LAST ELEMENT CREATED	0
ND - SIZE OF THE DYNAMIC STORAGE	OPTION X 4096
NF - FIRST AVAILABLE BLOCK	LOC(D(ND-1))
NL - LAST AVAILABLE BLOCK	LOC(D(ND-1))
D - DYNAMIC STORAGE, D(I),I=1,..,ND	D(ND-1)=0

INITIALLY, THE ENTIRE DYNAMIC STORAGE AREA CAN BE VIEWED AS A LARGE UNUSED BLOCK. WHENEVER A NEW UNIVERSAL ELEMENT IS CREATED BY THE EXECUTION OF 'CREATE' STATEMENTS, ONE COMPUTER WORD IS CARVED OUT FROM THE TOP OF THIS BLOCK AND THE POINTER NU IS INCREMENTED BY 1. THE CREATION OF NEW ELEMENTS MAYBE CONTINUED UNTIL THE LOC(D(NU)) IS EQUAL TO NL. (I.E. THE AVAILABLE SPACE IS EXHAUSTED) THE ALLOCATION OF 2-WORD BLOCKS IS CARRIED OUT BY TAKING AWAY THE BOTTOM TWO WORDS FROM THE UNUSED BLOCK. WHENEVER A BLOCK IS RELEASED (I.E. AVAILABLE FOR OTHER USES) IT IS LINKED TOGETHER TO FORM A SINGLELY-LINKED AVAILABLE-BLOCK LIST WITH NF AS ITS LIST HEADER. THE SUBSEQUENT REQUESTS FOR A 2-WORD BLOCK WILL BE SATISFIED THROUGH THE ALLOCATION OF A BLOCK FROM THIS AVAILABLE-BLOCK LIST. THE NEW BLOCK WILL NOT BE CREATED OUT OF THE UNUSED PORTION OF THE DYNAMIC STORAGE AREA UNTIL THE AVAILABLE-BLOCK LIST IS EXHAUSTED. IN OTHER WORDS, THE STORAGE MANAGEMENT ROUTINE TRIES TO SATISFY THE REQUEST FOR A 2-WORD BLOCK BY ASSIGNING AN EXISTING AVAILABLE BLOCK BEFORE AN ATTEMPT IS MADE TO ALLOCATE A NEW BLOCK. THE ALGORITHM FOR ALLOCATING A 2-WORD BLOCK (WHICH IS TO BE POINTED TO BY 'AVAIL') IS AS FOLLOWS : (NOTE : LINK(A) DENOTES THE CONTENTS OF THE LINK FIELD OF VARIABLE A)

```
1.  THE AVAILABLE-BLOCK LIST IS NOT EMPTY
        IF ( NF = 0 )  THEN GO TO 2.
          ELSE
               AVAIL     = NF
               LINK(NF)  = LINK(LINK(NF))
               RETURN

2.  THE AVAILABLE-BLOCK LIST IS EMPTY, CHECK THE UNUSED PORTION
        IF ( NL < LOC(D(NU))+2 )  THEN GO TO 3.
          ELSE
               AVAIL     = NF
               LINK(NF)  = LOC(NF)-2
               NL        = LINK(NF)
               RETURN

3.  THE AVAILABLE SPACE IS EXHAUSTED
        OUTPUT MESSAGE
```

TERMINATE PROGRAM

THE ALGORITHM TO FREE A 2-WORD BLOCK (D(K), D(K+1)) IS :

1. LINK THE 2-WORD BLOCK INTO THE AVAILABLE-BLOCK LIST
   LINK(D(K)) = LINK(NF)

2. UPDATE THE LIST HEADER, NF
   LINK(NF) = LOC(D(K))
   RETURN

THE SCHEMATIC DIAGRAM OF THE DYNAMIC STORAGE AREA IS GIVEN IN FIGURE 1. (K1,K2,...,KM ARE BLOCKS IN THE AVAILABLE-BLOCK LIST).

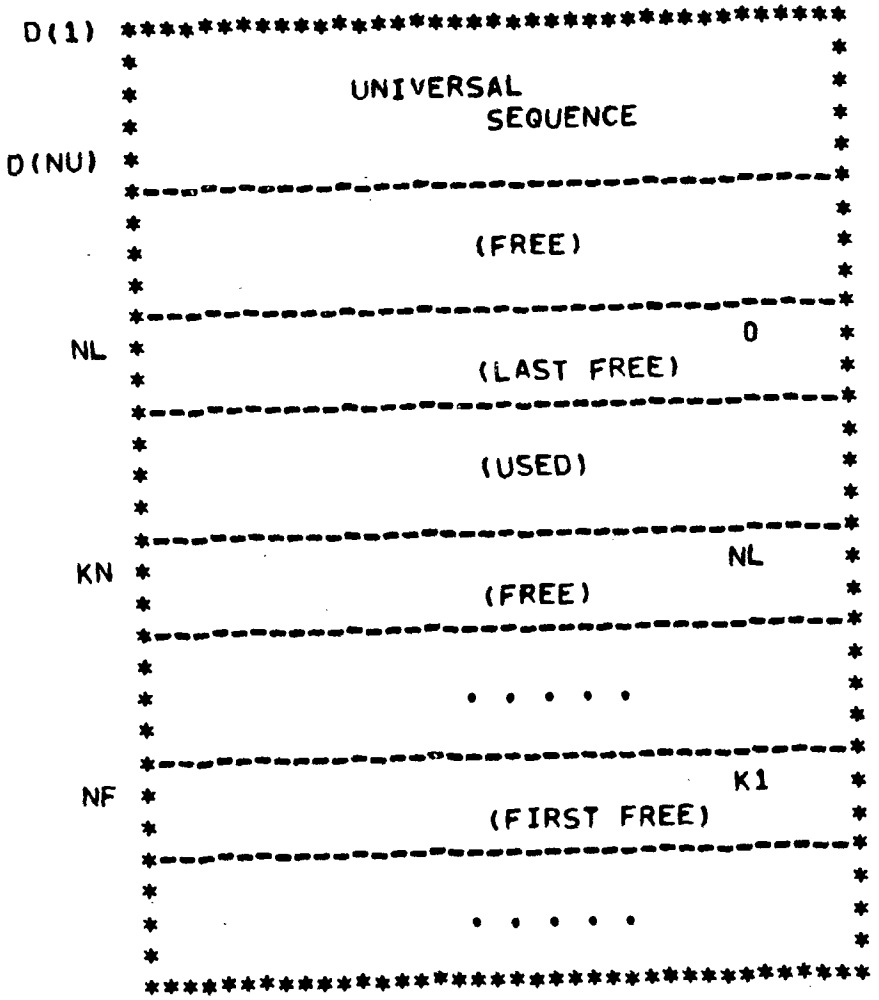


FIGURE 1. DYNAMIC STORAGE AREA.

3.2, DATA REPRESENTATION

ALL LINKS BETWEEN THE BLOCKS REPRESENTING DATA ARE WITH ABSOLUTE ADDRESSES EXCEPT REFERENCES TO THE ELEMENTS IN THE UNIVERSAL SEQUENCE WHERE THEIR RELATIVE ADDRESSES ARE USED, E.G. 'I' REFERS TO THE ELEMENT IN THE UNIVERSAL SEQUENCE D(I).

3.2.1. UNIVERSAL SEQUENCE AND THE PROPERTY BLOCKS

THE UNIVERSAL SEQUENCE IS A SEQUENCE OF ELEMENTS GENERATED BY THE FGRAAL 'CREATE' FUNCTIONS. IT IS THE UNION OF ALL SET ELEMENTS IN THE FGRAAL SYSTEM (E.G. NODES(G), ARCS(G), ELEMENTS OF ANY SET), HENCE IT IS CALLED THE UNIVERSAL SEQUENCE. THIS SEQUENCE IS IMPLEMENTED AS A LINEAR ARRAY LOCATED ON THE UPPER END OF THE DYNAMIC STORAGE AREA, D(I), I=1,2,...,NU. EACH MEMORY WORD IN THIS ARRAY, CALLED AN ELEMENT ID WORD, REPRESENTS ONE ELEMENT IN THE UNIVERSAL SEQUENCE. THE RELATIVE LOCATION OF AN ELEMENT IN THIS SEQUENCE IS CALLED ITS UNIVERSAL SEQUENCE NUMBER. DIFFERENT TYPES OF PROPERTIES MAYBE ASSIGNED TO AN ATOMIC SET (HENCE TO ITS ONLY ELEMENT IN THE SET) BY THE PROPERTY ASSIGNMENT OR CONDITIONAL 'CREATE' STATEMENTS IN FGRAAL. THE PROPERTIES ASSOCIATED WITH AN ELEMENT ARE IMPLEMENTED AS A PROPERTY-RING WHERE ITS ELEMENT ID WORD IS A RING HEADER AND PROPERTY BLOCKS ARE THE RING ELEMENTS.

EACH ELEMENT ID WORD IS DIVIDED INTO TWO HALFWORD FIELDS. THE LEFT HALF OF THE ELEMENT ID WORD IS RESERVED FOR THE REPRESENTATION OF SETS IN COLUMN FORM (SEE SETS). THE RIGHT HALF OF THE ELEMENT ID WORD CONTAINS THE ADDRESS POINTER TO A LINKED LIST OF PROPERTY BLOCKS WHICH IS ASSOCIATED WITH THIS ELEMENT (IF THERE IS ONE), OR THE COMPLEMENT OF ITS UNIVERSAL SEQUENCE NUMBER TO SIGNIFY THE NONEXISTENCE OF THE PROPERTY LIST. THE LINK FIELD OF THE LAST PROPERTY BLOCK ON THE PROPERTY LIST CONTAINS THE COMPLEMENT OF THE UNIVERSAL SEQUENCE NUMBER (WHICH POINTS INDIRECTLY TO THE ELEMENT ID WORD). THE COMPLEMENT IS NECESSARY TO SIGNIFY THE END OF A PROPERTY LIST. THE ELEMENT ID WORD TOGETHER WITH ITS ASSOCIATED PROPERTY-LIST FORM A PROPERTY-RING WHERE ELEMENT ID WORD IS ITS RING HEADER. FIGURE 3 SHOWS THE LAYOUT OF THE UNIVERSAL SEQUENCE AND THE PROPERTY BLOCK LISTS.

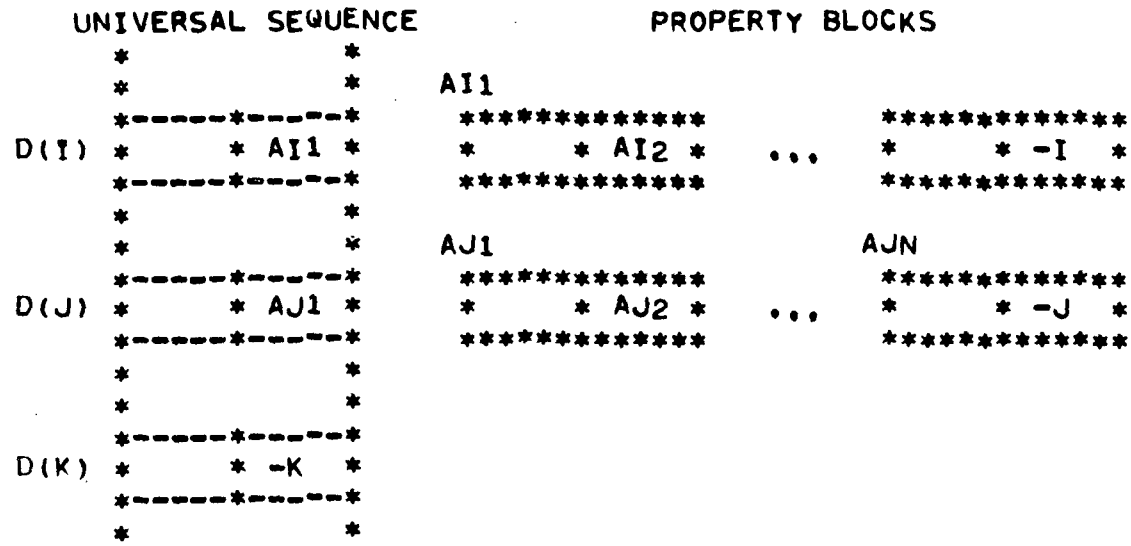


FIGURE 3. UNIVERSAL SEQUENCE AND PROPERTY BLOCKS

3.2.2. SETS

IN THE FGRAAL SYSTEM, THERE ARE THREE CATEGORIES OF SETS : EMPTY, ATOMIC, AND COMPOSITE. AN EMPTY SET IS A SET WHICH CONTAINS NO ELEMENTS. AN ATOMIC SET IS A SET WHICH CONTAINS EXACTLY ONE ELEMENT. A COMPOSITE SET IS A SET WHICH IS THE UNION OF A NUMBER OF ATOMIC SETS. SETS, REGARDLESS OF THEIR CATEGORIES, ARE ALWAYS REPRESENTED WITH A SET HEADER. IF A SET IS EMPTY OR ATOMIC, IT WILL BE REPRESENTED BY A SET HEADER ALONE. HOWEVER, IF A SET IS A COMPOSITE SET THEN IT WILL BE REPRESENTED BY A SET HEADER TOGETHER WITH A LIST OF SET ELEMENTS. THERE ARE TWO WAYS OF REPRESENTING A LIST OF SET ELEMENTS : (1) LINKED-BLOCK REPRESENTATION AND (2) COLUMN REPRESENTATION. A SET WHOSE ELEMENTS ARE REPRESENTED IN THE LINKED-BLOCK FORM IS SOMETIMES CALLED BLOCK SET. A SET REPRESENTED IN COLUMN FORM IS SOMETIMES CALLED COLUMN SET.

THE SET HEADER OF A DECLARED SET IS A MEMORY WORD ASSIGNED BY THE COMPILER, WHICH IS LOCAL TO THE SUBPROGRAM WHERE IT HAD BEEN DECLARED. DURING THE PROGRAM EXECUTION, THE LOCATION OF THIS ASSIGNED MEMORY WORD (OR SET HEADER) BECOMES THE UNIQUE IDENTIFIER FOR THE ASSOCIATED, DECLARED SET. THE CONTENTS OF THE SET HEADER IDENTIFIES ITS ASSOCIATED SET AS : EMPTY, ATOMIC, OR COMPOSITE. FOR A COMPOSITE SET, THERE IS A LIST OF SET ELEMENTS ASSOCIATED WITH ITS SET HEADER. THE LINKED-BLOCK REPRESENTATION CONSISTS OF SINGLY LINKED 2-WORD BLOCKS WHICH CONTAIN THE SET ELEMENTS. THE COLUMN REPRESENTATION CONSISTS OF THREE MAJOR COMPONENTS : (1) ASSIGNMENT OF A COLUMN POSITION IN THE UNIVERSAL SEQUENCE TO THIS SET, (2) SETTING OF THE ASSIGNED COLUMN POSITION IN THE UNIVERSAL ELEMENT TO INDICATE THE ELEMENT IS ALSO CONTAINED IN THIS SET, AND (3) SUPPLEMENTARY INFORMATION CONCERNING THIS SET (E.G. NO. OF ELEMENTS, FIRST AND LAST ELEMENT, ETC.). IF A SET IS EMPTY OR ATOMIC THEN IT IS SUFFICIENT TO REPRESENT THE SET SOLELY BY ITS ASSOCIATED SET HEADER. THE REPRESENTATIONS OF THESE SETS ARE GIVEN IN FIGURE 4. THE EMPTY SET IS REPRESENTED BY A SET HEADER CONTAINING ALL 0'S. THE ATOMIC SET WITH THE ELEMENT WHOSE UNIVERSAL SEQUENCE NUMBER IS I, IS REPRESENTED BY A SET HEADER WHICH CONTAINS THE NUMBER I IN ITS RIGHT HALF.

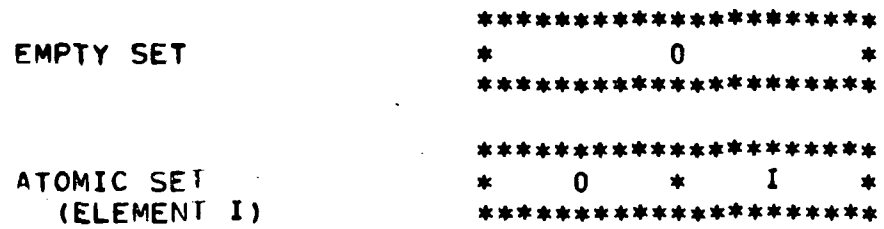


FIGURE 4. EMPTY AND ATOMIC SET

FOR A COMPOSITE SET, THE SET HEADER IS DIVIDED INTO TWO HALF-WORD FIELDS. THE LEFT HALF CONTAINS THE ADDRESS POINTER TO THE LIST OF SET ELEMENTS BELONGING TO THIS SET, WHILE THE RIGHT HALF CONTAINS 0'S. THE LIST OF SET ELEMENTS REPRESENTED IN LINKED-BLOCK FORM ARE ARRANGED IN A SINGLELY-LINKED LIST CONSISTING OF 2-WORD BLOCKS. EACH 2-WORD BLOCK IS SUBDIVIDED INTO 4 HALFWORD FIELDS. THE SECOND HALFWORD FIELD (OR THE RIGHT HALF OF THE FIRST WORD) IS THE LINK FIELD WHICH CONTAINS ADDRESS POINTER TO THE NEXT 2-WORD BLOCK (IF THERE IS ONE) OR IT CONTAINS 0'S, IF THIS BLOCK IS THE LAST ONE ON THE LINKED LIST. THE REMAINING THREE FIELDS OF EACH BLOCK, CALLED ELEMENT FIELDS, CONTAIN SET ELEMENTS. THE SET ELEMENTS ARE ARRANGED IN THE ORDER OF THEIR UNIVERSAL SEQUENCE NUMBER, THE ELEMENTS WITH SMALLER UNIVERSAL SEQUENCE NUMBERS ARE STORED FIRST. THE SEQUENCE IN WHICH THE ELEMENT FIELDS ARE UTILIZED IS : FIRST,THIRD, FOURTH (OR EQUIVALENTLY, LEFT HALF OF THE FIRST WORD, LEFT HALF OF THE SECOND WORD, AND RIGHT HALF OF THE SECOND WORD). WHEN THE NUMBER OF ELEMENTS IN A SET IS NOT DIVISIBLE BY THREE, THE LAST BLOCK WILL CONTAIN UNUSED ELEMENT FIELD(S). THE UNUSED ELEMENT FIELDS ARE ZERO TO DENOTE THEIR EMPTINESS. AN EXAMPLE OF A BLOCK SET IS SHOWN IN FIGURE 5.

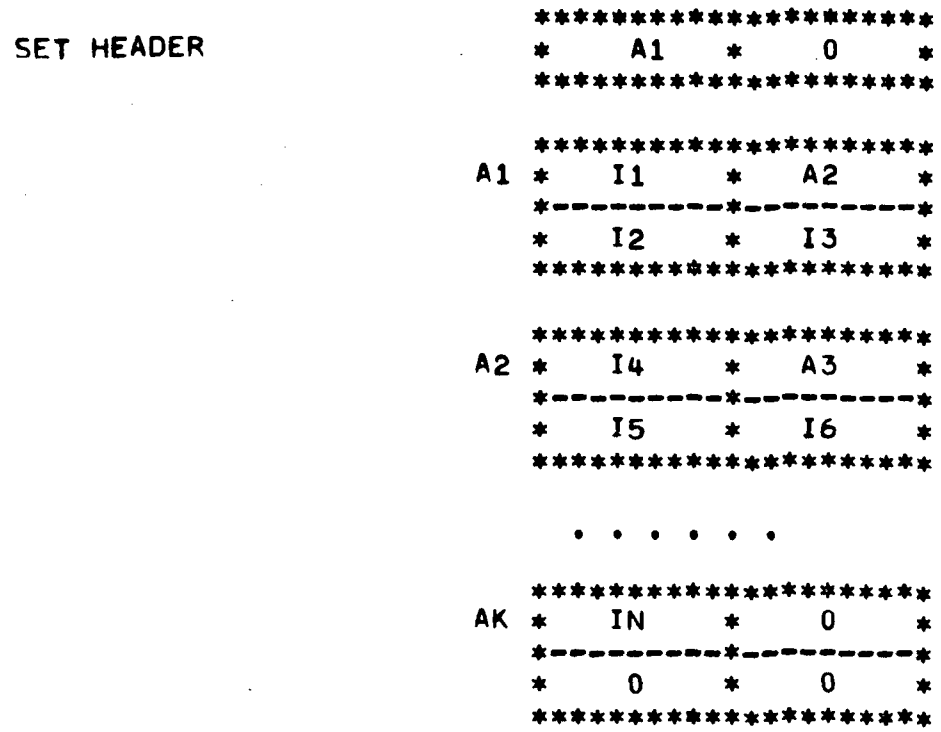


FIGURE 5. SET IN BLOCK FORM CONTAINING ELEMENTS
I1 < I2 < ... < IN

IF THE LIST OF SET ELEMENTS IS REPRESENTED IN COLUMN FORM, THE LOCATION POINTED TO BY ITS SET HEADER IS A 2-WORD BLOCK, CALLED THE SET INFOR BLOCK, WHICH CONTAINS INFORMATION CONCERNING ITS SET ELEMENTS. THE INFOR BLOCK IS SUBDIVIDED INTO 4 HALFWORD FIELDS : (1) COUNT, (2) COLUMN INDEX, (3) FIRST AND (4) LAST. THE COUNT FIELD CONTAINS THE COMPLEMENT OF THE NUMBER OF TOTAL ELEMENTS IN THE SET. THE COMPLEMENT OF THE COUNT IS NECESSARY FOR THE PURPOSE OF DISTINGUISHING A COLUMN FORM REPRESENTATION FROM THE LINKED-BLOCK FORM REPRESENTATION. THE COLUMN INDEX FIELD CONTAINS AN INTEGER K, 0 < K < 16, WHICH INDICATES THE ASSIGNMENT OF COLUMN K IN THE UNIVERSAL SEQUENCE TO THIS SET. THE RIGHTMOST COLUMN OF THE LEFT HALF OF THE UNIVERSAL SEQUENCE IS DENOTED AS COLUMN 1, WHILE THE LEFTMOST COLUMN (OR SIGN BIT) IS DENOTED AS COLUMN 18 (NOTE: THE LEFTMOST THREE COLUMN POSITIONS ARE NOT USED FOR REPRESENTING SETS). THE UNIVERSAL ELEMENT HAS ITS K,TH COLUMN SET TO 1 IF AND ONLY IF IT ALSO BELONGS TO THE SET WHICH HAS COLUMN K ASSIGNED TO IT. THE FIRST AND LAST FIELDS CONTAIN THE UNIVERSAL SEQUENCE NUMBER OF THE FIRST AND LAST ELEMENT OF THE SET, RESPECTIVELY.

FOR EACH COLUMN K (0<K<16) THERE ARE TWO INFORMATION WORDS, C1(K) AND C2(K), ASSOCIATED WITH IT. THE ARRAYS C1 AND C2 CONTAIN INFORMATION FOR MANAGING THESE 15 COLUMNS. IF COLUMN K IS NOT ASSIGNED TO ANY SET, THE RIGHT HALF OF C1(K) CONTAINS A COLUMN NUMBER WHICH IS THE NEXT FREE (UNASSIGNED) COLUMN, OR IT CONTAINS 0'S TO INDICATE THAT THIS IS THE LAST FREE COLUMN. THIS IS, IN FACT, A LINKED LIST OF AVAILABLE COLUMN SPACES. HOWEVER, IF COLUMN K IS CURRENTLY ASSIGNED TO A SET, THEN THESE TWO WORD CONTAIN INFORMATIONS AS FOLLOWS : (1) LEFT HALF OF C1(K) CONTAINS AN ADDRESS POINTER TO THE 'INFOR BLOCK', (2) RIGHT HALF OF C1(K), CALLED TIME FIELD, CONTAINS A GLOBAL REFERENCE COUNT VALUE WHICH RANKS SETS BY THEIR LATEST USE, AND (3) C2(K) IS SUBDIVIDED INTO SIX SIXTHWORD FIELDS. THE FIRST SIXTHWORD OF C2(K) IS THE USE-TAG FIELD WHICH IS SET TO ONE WHENEVER THE SET IS USED IN THE CURRENT OPERATION. THE SECOND SIXTHWORD FIELD IS THE PROPERTY-TAG FIELD WHICH IS SET TO ONE IF THE SET IS REPRESENTING NODES(G), ARCS(G), OR DOMAIN(P). THE REMAINING 4 FIELDS OF C2(K) ARE NOT USED AND ARE AVAILABLE FOR FUTURE EXPANSION. AN EXAMPLE OF A COLUMN SET IS SHOWN IN FIGURE 6.

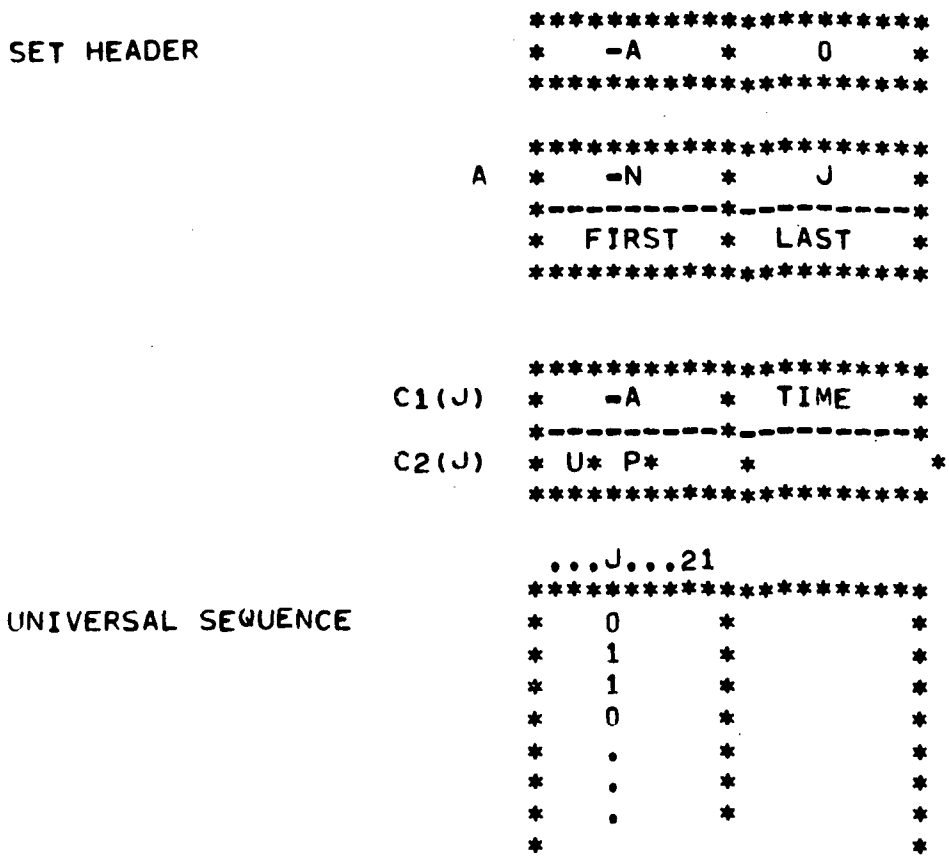


FIGURE 6. SET IN COLUMN FORM

THE RESULTS OF SET OPERATIONS MUST BE STORED IN COLUMN FORM (IN ORDER TO SPEED UP THE SET OPERATION). SINCE A TOTAL OF 15 COLUMN SETS MAY EXIST IN THE SYSTEM AT ANY ONE GIVEN TIME, A MANAGEMENT SCHEME IS REQUIRED TO TRANSFER SETS FROM COLUMN FORM TO BLOCK FORM IN ORDER TO MAKE COLUMN SPACE AVAILABLE. TO PREVENT THE TRANSFER OF A COLUMN SET WHICH IS CURRENTLY BEING USED THE USE-TAG FILED ASSOCIATED WITH THIS COLUMN SET IS CHECKED. TO PREVENT THE TRANSFER OF THE SET OF NODES(G) OR ARCS(G) TO A BLOCK FORM (WHICH WOULD PRODUCE A SECOND LINKED LIST COPY OF THIS SET WHICH ALREADY EXISTS IN PROPERTY BLOCKS) THE PROPERTY-TAG FIELD ASSOCIATED WITH THIS COLUMN SET IS CHECKED.

DURING THE PROGRAM EXECUTION, A GLOBAL COUNTER IS MAINTAINED (FOR COLUMN SETS REFERENCES). WHENEVER ANY COLUMN SET (SAY WITH COLUMN INDEX K) IS REFERENCED (E.G. DURING SET OPERATIONS) THE GLOBAL COUNTER IS INCREMENTED BY ONE AND ITS UPDATED VALUE IS STORED INTO THE RIGHT HALF OF C1(K) (THE TIME FIELD). IF A RE-

QUEST FOR A COLUMN IS RECOGNIZED AND THERE IS NO FREE COLUMN AVAILABLE, THE COLUMN MANAGEMENT ROUTINE WILL SELECT ONE COLUMN SET AND TRANSFER IT TO BLOCK FORM. THE COLUMN SET WITH MINIMUM VALUE IN THE TIME FIELD AND WHOSE USE-TAG IS NOT SET TO ONE WILL BE SELECTED. THE COLUMN NUMBER ASSOCIATED WITH THIS SELECTED SET WILL THEN BE USED TO SATISFY THE OUTSTANDING COLUMN REQUEST.

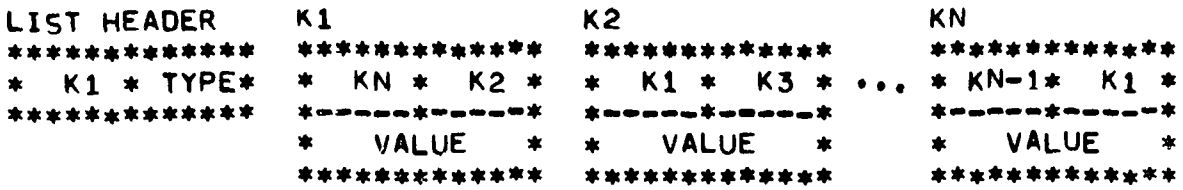
3.2.3. STAQUES (LISTS)

THE STAQUE (OR LIST) IS ONE OF THE NEW DATA STRUCTURES INTRODUCED IN THE FGRAAL. FROM THE STORAGE STRUCTURE POINT OF VIEW THERE ARE THREE DIFFERENT CATEGORIES OF STAQUES : (1) SINGLE WORD -LOGICAL, INTEGER AND REAL TYPES OF STAQUE , (2) DOUBLE WORD -- DOUBLE PRECISION AND COMPLEX TYPES OF STAQUE, AND (3) VARIABLE SIZE -- SET TYPE OF STAQUE. STAQUES, REGARDLESS OF THEIR DATA TYPES, ARE ALWAYS REPRESENTED BY A STAQUE HEADER AND A CIRCULAR DOUBLY-LINKED LIST OF ITS STAQUE ELEMENTS.

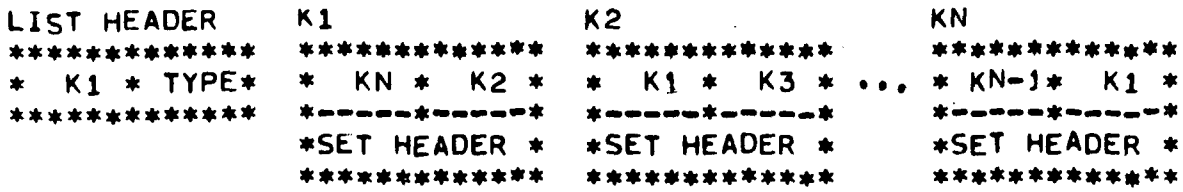
THE STAQUE HEADER OF A DECLARED STAQUE IS A MEMORY WORD ASSIGNED BY THE COMPILER, WHICH IS LOCAL TO THE SUBPROGRAM WHERE IT HAD BEEN DECLARED. DURING THE PROGRAM EXECUTION, THE ADDRESS OF THIS MEMORY WORD BECOMES THE UNIQUE IDENTIFIER OF THE ASSOCIATED STAQUE. THE STAQUE HEADER CONTAINS TWO HALFWORD FIELDS. THE LEFT HALF OF THE STAQUE HEADER CONTAINS THE ADDRESS POINTER TO THE LINKED LIST OF ITS STAQUE ELEMENTS. THE RIGHT HALF OF THE HEADER IS THE TYPE FIELD WHICH CONTAINS THE DATA TYPE IDENTIFIER OF ITS ASSOCIATED STAQUE.

THE STAQUE ELEMENTS ARE ARRANGED IN THE FORM OF A CIRCULAR DOUBLY-LINKED LIST CONSISTING OF 2-WORD BLOCKS. THE FIRST WORD OF THE 2-WORD BLOCK IS DIVIDED INTO TWO HALFWORD LINK FIELDS. THE LEFT HALF IS THE BACKWARD LINK FIELD POINTING TO THE PREVIOUS BLOCK AND THE RIGHT HALF IS THE FORWARD LINK FIELD POINTING TO THE NEXT BLOCK. THE FIRST AND LAST BLOCKS OF THE STAQUE ELEMENT LIST ARE LINKED TOGETHER CIRCULARLY (I.E. THE BACKWARD POINTER OF THE FIRST POINTS TO THE LAST AND THE FORWARD POINTER OF THE LAST POINTS TO THE FIRST). THE SECOND WORD OF THE 2-WORD BLOCK CONTAINS DIFFERENT INFORMATION DEPENDING ON THE DATA TYPE OF ITS STAQUE. FOR THE DATA TYPES IN THE SINGLE WORD CATEGORY, IT CONTAINS THE DATA VALUE OF THE ELEMENT. FOR THE SET DATA TYPE, IT CONTAINS THE SET HEADER. FOR THE DATA TYPES IN THE DOUBLE WORD CATEGORY, IT CONTAINS THE ADDRESS POINTER TO ANOTHER 2-WORD BLOCK WHICH, IN TURN, CONTAINS THE DATA VALUES OF THE ELEMENT. THE REPRESENTATION OF DIFFERENT CATEGORIES OF THE STAQUES ARE GIVEN IN FIGURE 7.

LOGICAL, INTEGER, REAL TYPES OF LISTS:



SET TYPE OF LISTS:



THE SET HEADERS ARE SIMILAR TO THE SET HEADERS OF THE NORMAL SETS EXCEPT NO COLUMN FORM IS USED.

DOUBLE PRECISION AND COMPLEX TYPES OF LISTS:

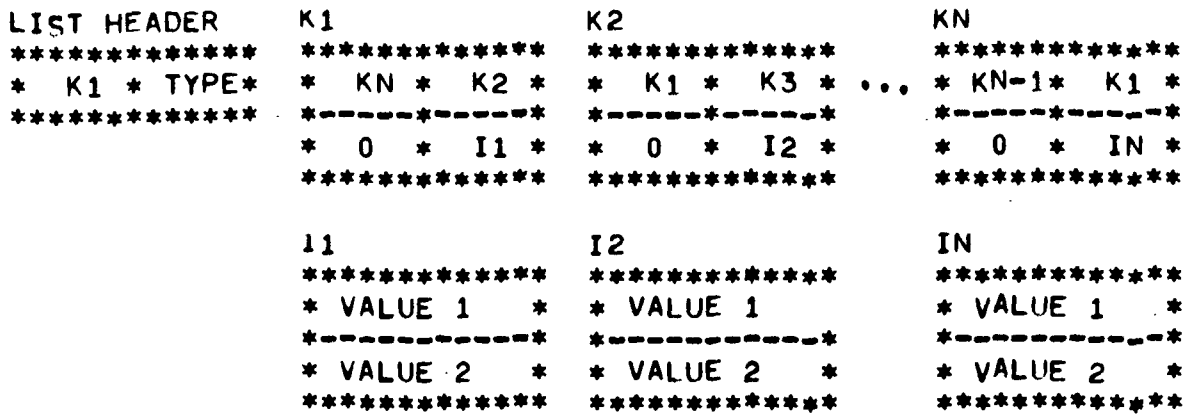


FIGURE 7. REPRESENTATIONS OF STAQUES

3.2.4. PROPERTIES

THE PROPERTY IS ANOTHER NEW DATA STRUCTURE INTRODUCED IN THE FGRAAL SYSTEM. FROM THE STORAGE STRUCTURE POINT OF VIEW THERE ARE THREE DIFFERENT CATEGORIES OF PROPERTIES (AS IT IS IN THE STAQUE): (1) SINGLE WORD -- LOGICAL, INTEGER AND REAL TYPES OF PROPERTY, (2) DOUBLE WORD -- DOUBLE PRECISION AND COMPLEX TYPES OF PROPERTY, AND (3) VARIABLE SIZE -- SET TYPE OF PROPERTY. PROPERTIES, REGARDLESS OF THEIR DATA TYPES, ARE ALWAYS REPRESENTED BY A PROPERTY HEADER, PROPERTY INFOR BLOCK AND PROPERTY VALUE BLOCK(S).

THE PROPERTY HEADER OF A DECLARED PROPERTY IS A MEMORY WORD ASSIGNED BY THE COMPILER (AS IT IS WITH THE SET HEADER). DURING THE PROGRAM EXECUTION, THE ABSOLUTE ADDRESS OF THIS ASSIGNED MEMORY WORD BECOMES THE UNIQUE IDENTIFIER OF THE ASSOCIATED PROPERTY. THE PROPERTY HEADER CONTAINS TWO HALFWORD FIELDS. THE RIGHT HALF OF THE PROPERTY HEADER IS THE TYPE FIELD WHICH CONTAINS THE DATA TYPE IDENTIFIER OF ITS ASSOCIATED PROPERTY. THE LEFT HALF OF THE PROPERTY HEADER IS THE POINTER FIELD WHICH CONTAINS THE COMPLEMENT OF THE ABSOLUTE ADDRESS OF A 2-WORD BLOCK CALLED PROPERTY INFOR BLOCK. THE COMPLEMENT OF THE ADDRESS IS USED IN ORDER TO DISTINGUISH A PROPERTY HEADER FROM A SET HEADER. THE PROPERTY INFOR BLOCK POINTED TO BY THE PROPERTY HEADER HAS THE SAME FORMAT AS THAT OF A COLUMN SET INFOR BLOCK (SEE SETS), EXECEPT THAT THE COLUMN INDEX FIELD IS ZERO UNLESS THE DOMAIN SET IS REPRESENTED BY A COLUMN SET.

THE ASSIGNMENT OF PROPERTIES TO AN ATOMIC SET CAN BE REALIZED THROUGH THE EXECUTION OF PROPERTY ASSIGNMENT OR CONDITIONAL 'CREATE' STATEMENTS. THE PROPERTY VALUE(S) THUS ASSIGNED TO AN (UNIVERSAL) ELEMENT IS STORED IN A PROPERTY BLOCK WHICH IS LINKED INTO THE PROPERTY-RING ASSOCIATED WITH THIS ELEMENT (SEE UNIVERSAL SEQUENCE AND PROPERTY BLOCKS). THE PROPERTY BLOCK CONSISTS OF TWO CONTIGUOUS MEMORY WORDS. THE FIRST MEMORY WORD IS SUBDIVIDED INTO TWO HALFWORD FIELDS. THE LEFT HALF IS THE PROPERTY IDENTIFICATION FIELD WHICH CONTAINS THE UNIQUE IDENTIFIER, THE ADDRESS OF THE PROPERTY HEADER, OF THE PROPERTY WITH WHICH THIS BLOCK IS ASSOCIATED. THE RIGHT HALF OF THE FIRST WORD IS THE LINK FIELD WHICH IS USED IN THE LINKAGE OF THE PROPERTY-RING (SEE UNIVERSAL SEQUENCE AND PROPERTY BLOCK). THE SECOND WORD OF THE PROPERTY BLOCK CONTAINS DIFFERENT INFORMATION DEPENDING ON THE DATA TYPE OF THE PROPERTY (THIS IS IN THE SAME FORMAT AS THAT OF THE STAQUE). FOR THE DATA TYPES IN THE SINGLE WORD CATEGORY, IT CONTAINS THE DATA VALUE OF THE PROPERTY. FOR THE SET DATA TYPE, IT CONTAINS THE SET HEADER. FOR THE DATA TYPES IN THE DOUBLE WORD CATEGORY, IT CONTAINS THE ADDRESS POINTER TO ANOTHER 2-WORD BLOCK WHICH, IN TURN, CONTAINS THE DATA VALUES OF THE PROPERTY. THE REPRESENTATIONS OF PROPERTIES ARE GIVEN IN FIGURE 8.

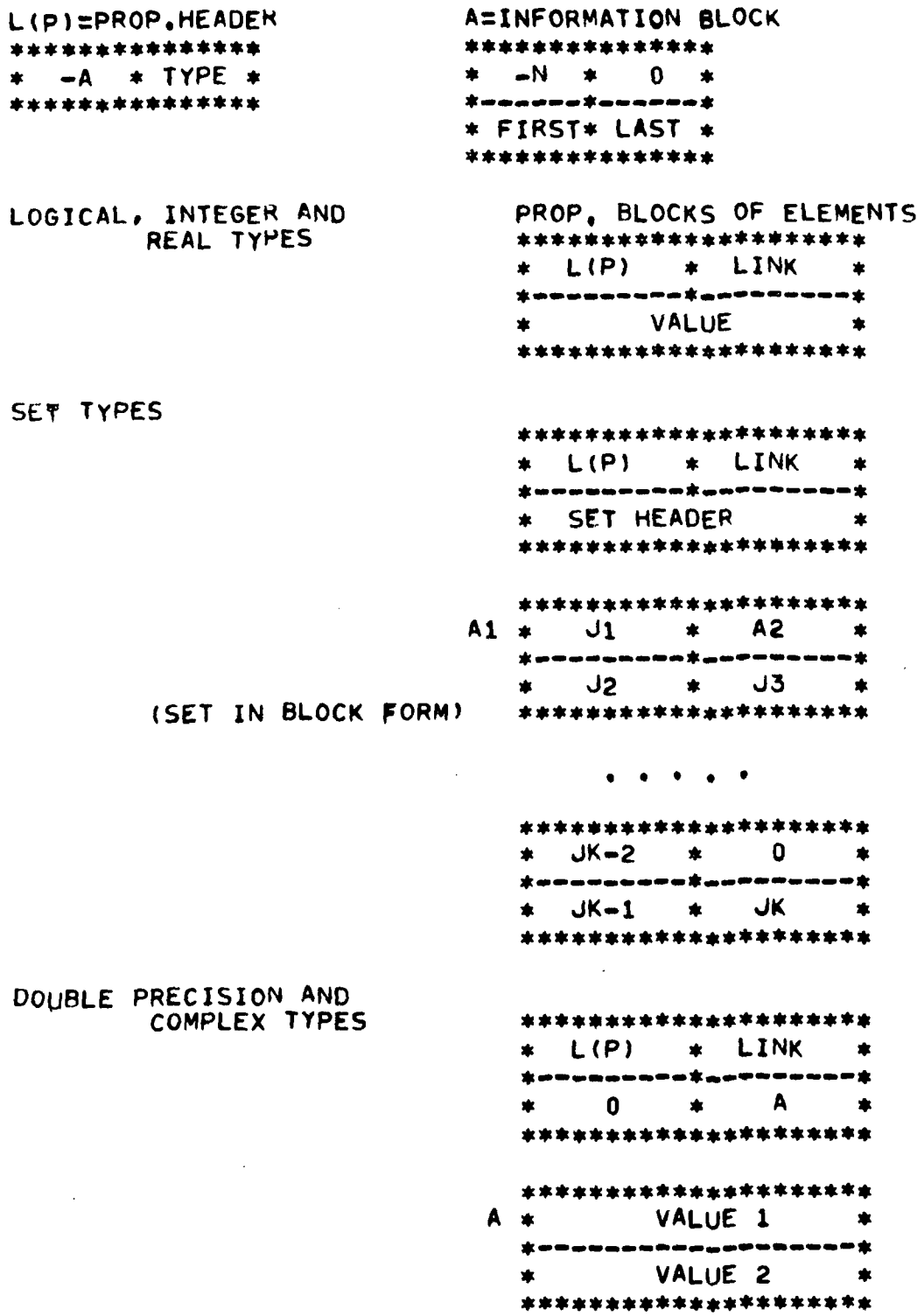
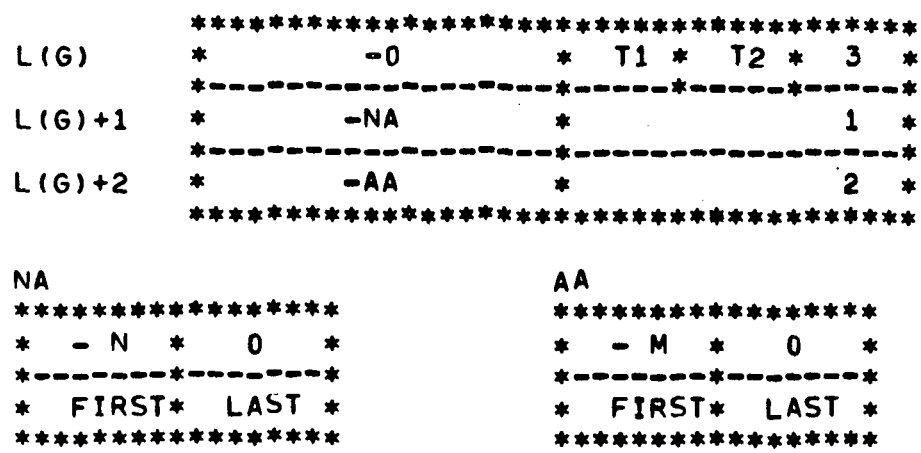


FIGURE 8. REPRESENTATION OF PROPERTY BLOCKS.

3.2.5. GRAPH STRUCTURES

GRAPH STRUCTURES ARE HANDLED SIMILARLY TO PROPERTIES. WITH EACH DECLARED GRAPH, A GRAPH HEADER CONSISTING OF 2 OR 3 WORDS IS ASSIGNED BY THE COMPILER. THE FIRST OF THESE WORDS IS THE GENERAL HEADER OF THE GRAPH, THE SECOND AND THIRD WORDS ARE THE HEADERS FOR THE NODES AND ARCS OF THE GRAPH, RESPECTIVELY. WHEN AN ELEMENT IS ASSIGNED TO THE GRAPH AS A NODE THEN IT HAS A PROPERTY BLOCK WHICH CONTAINS THE ADDRESS OF THE NODE HEADER OF THE GRAPH AS PROPERTY IDENTIFIER. IF AN ELEMENT IS AN ARC THEN IT USES THE ADDRESS OF THE ARC HEADER OF THE GRAPH. AN ELEMENT MAY NOT BE ASSIGNED TO THE SAME GRAPH AS BOTH NODE AND ARC.



T1 = 0 FOR NODE GRAPH
1 FOR NODE/ARC GRAPH
N = NUMBER OF NODES
M = NUMBER OF ARCS
T2 = 0 FOR UNDIRECTED GRAPH
1 FOR DIRECTED GRAPH
THE INTEGERS 3,1 AND 2 IN L(G),L(G)+1 AND L(G)+2, RESPECTIVELY, ARE USED AS IDENTIFIERS.

FIGURE 9. GRAPH HEADER WITH NODE AND ARC INFORMATION BLOCKS

THE PROPERTY BLOCKS ASSIGNED TO THE ELEMENTS AS NODES OR ARCS OF THE GRAPH CONSIST OF ONE BLOCK (2 WORDS). THE LEFT HALF OF THE FIRST WORD CONTAINS THE PROPERTY IDENTIFIER, I.E. THE ADDRESS OF THE NODE OR ARC HEADER. THE RIGHT HALF OF THE FIRST WORD CONTAINS THE LINK TO THE NEXT PROPERTY BLOCK. THE SECOND WORD OF THE PROPERTY BLOCK CONTAINS THE STRUCTURAL INFORMATION. THE STRUCTURAL INFORMATION OF THE GRAPH IS STORED BY ABSOLUTE ADDRESS LINKS. FOR EACH NODE OF A GRAPH, THE SETS CONSISTING OF THE NEGATIVE COBOUNDARY AND POSITIVE COBOUNDARY OF THE NODE IS REPRESENTED BY A LINK STRUCTURE. THE LEFT HALF OF THE SECOND WORD OF THE PROPERTY BLOCK IS USED TO LINK THE NEGATIVE, THE RIGHT HALF WORD IS USED TO LINK THE POSITIVE COBOUNDARY SETS. THE LINKAGE IS CIRCULAR, I.E. THE LAST ARC IS LINKED BACK TO THE NODE. IF A NODE HAS EMPTY COBOUNDARY (POSITIVE OR NEGATIVE) THEN IT IS LINKED TO ITSELF. UNDIRECTED GRAPHS USE THE DIRECTION USED IN THE ASSIGN STATEMENT. NODE GRAPHS USE INDEPENDENT 2-WORD BLOCKS FOR ARCS, I.E. THESE BLOCKS ARE NOT LINKED TO ELEMENTS IN THE UNIVERSAL SEQUENCE AS PROPERTY BLOCKS.

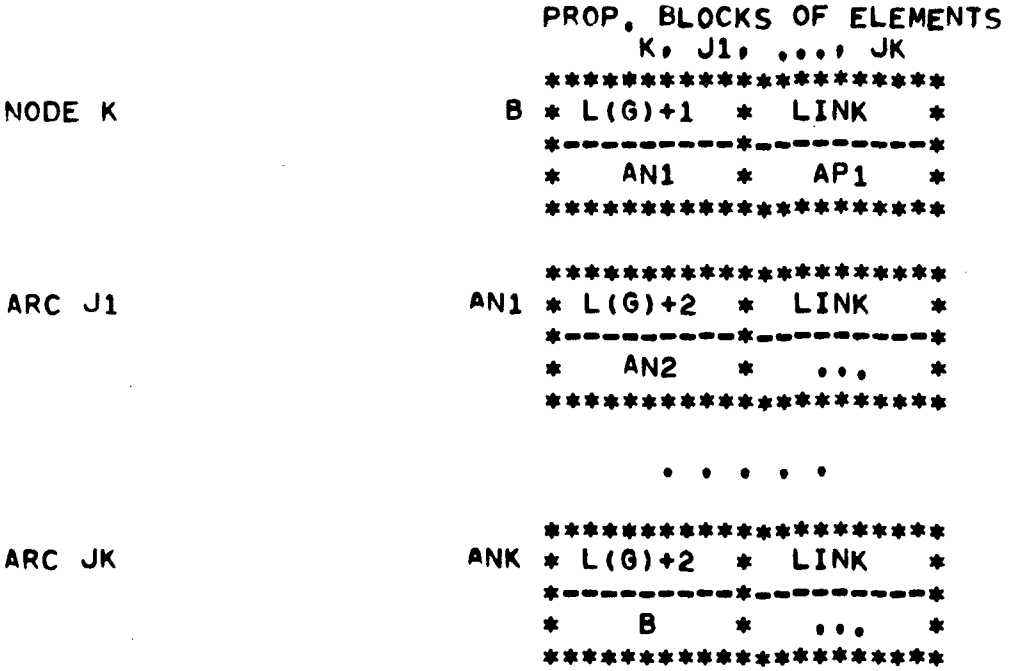


FIGURE 10. GRAPH STRUCTURE LINKAGE.
THE NEGATIVE COBOUNDARY OF NODE K.
L(G) = LOCATION OF THE GRAPH HEADER

3.2.6. TYPE ASSIGNMENTS.

THE HEADERS OF LISTS, PROPERTIES AND GRAPHS CONTAIN A FIELD CONSISTING OF A TYPE CODE. THIS TYPE CODE IS AN INTEGER AS FOLLOWS:

0	SET
1	NODES
2	ARCS
3	GRAPH
4	LOGICAL
5	INTEGER
6	REAL
7	DOUBLE PRECISION
8	COMPLEX

3.2.7. STRUCTURE OF HEADERS

THE FOLLOWING TABLE SUMMARIZES THE CONTENTS OF THE HEADERS:

	1111111	1122222222233333
	0123456789012345	8901234567890123 45
ELEMENT		
VALID	COLUMN SET BITS	ADDRESS OF PROP. BLOCK OR COMPL. OF THE ELEMENT
REMOVED	1000000000000000	
SET		
EMPTY	ZERO	ZERO
ATOMIC	ZERO	ELEMENT
BLOCK	ADDRESS OF THE FIRST BLOCK	ZERO
COLUMN	ADDRESS OF THE INFO. BLOCK	ZERO
LIST		
SET	ADDRESS	0
LOGICAL	OF	4
INTEGER	THE	5
REAL	FIRST	6
DBL.PR.	ELEMENT	7
COMPLEX	BLOCK	8
PROPERTY		
SET	-ADDRESS	0
LOGICAL	OF THE	4
INTEGER	PROPERTY	5
REAL	INFORMATION	6
DBL.PR.	BLOCK	7
COMPLEX		8
GRAPH		
GRAPH	-0	X Y 3
NODES	-ADDRESS OF INFO.	1
ARCS	BLOCK	2
INFO.BLOCK		
FIRST WORD	-NO. OF ELEMENTS	ZERO OR COLUMN NUMBER IF IN SET
SECOND W.	FIRST ELEMENT	LAST ELEMENT

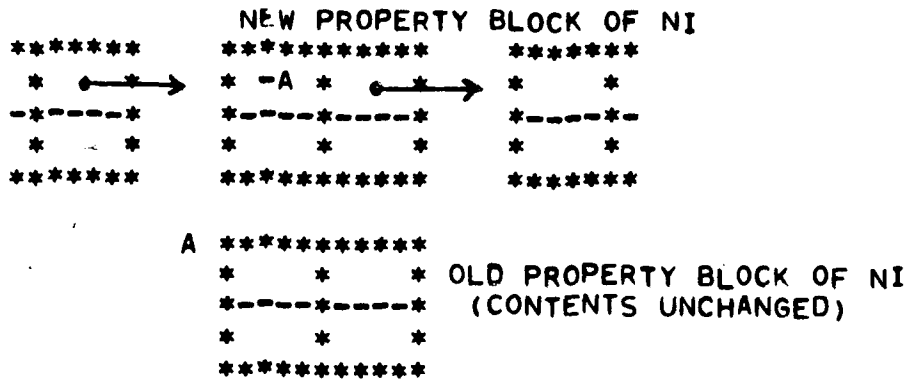
NOTES:
- X = ZERO OR ONE FOR NOUE OR NODE/ARC GRAPH, RESPECTIVELY.
- Y = ZERO OR ONE FOR UNDIRECTED OR DIRECTED GRAPH, RESP.
- - (MINUS) INDICATES COMPLEMENT IN THE PROPER HALF WORD.

3.3. REPRESENTATION OF A CONTRACTED GRAPH

CONTRACTION OF A SUBGRAPH CORRESPONDING TO THE GIVEN SET OF NODES IS ACCOMPLISHED IN TWO STEPS:

STEP 1.

A NEW NODE, N, IS CREATED WHICH WILL CORRESPOND TO THE CONTRACTED SUBGRAPH. ALL NODES INVOLVED IN THE CONTRACTION, INCLUDING THOSE NODES WHICH WERE PART OF PREVIOUS CONTRACTION, ARE ASSIGNED NEW PROPERTY BLOCKS. CALL THESE NODES N1,N2,...,NM. THE OLD PROPERTY BLOCKS ARE LINKED TO THE NEW BLOCKS BY STORING THEIR NEGATIVE ADDRESSES IN THE LEFT HALVES OF THE FIRST WORD OF THE NEW PROPERTY BLOCKS:



STEP 2.

THE ARCS IN EACH ORIGINAL POSITIVE AND NEGATIVE COBOUNDARY CYCLE ARE REARRANGED SO THAT THE ARCS INTERNAL TO THE CONTRACTED SUBGRAPH PRECEDE THE ARCS CONNECTING THE SUBGRAPH WITH THE REST OF THE GRAPH. THE LINKAGES IN THE SECOND WORD OF THE NEWLY CREATED PROPERTY BLOCKS, ASSOCIATED WITH THE NODES N,N1,...,NM ARE DEFINED AS FOLLOWS:

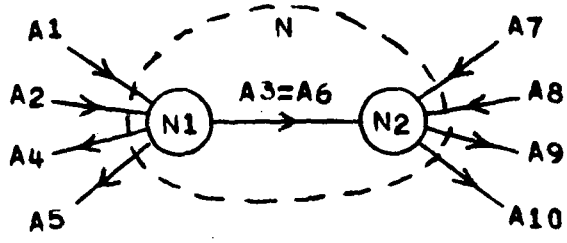
1. THE LEFT FIELD OF THE SECOND WORD (NCOB POINTER) OF N POINTS TO THE FIRST NON-INTERNAL ARC OF THE NCOB OF N1. IF NO NON-INTERNAL ARCS EXIST, THEN IT POINTS TO N1.

2. FOR NI ($1 \leq I \leq M-1$), THE LEFT FIELD OF THE SECOND WORD OF NI POINTS TO THE FIRST NON-INTERNAL ARC OF THE NCOB OF NI+1. IF NO NON-INTERNAL ARCS EXIST, THEN IT POINTS TO NI+1.

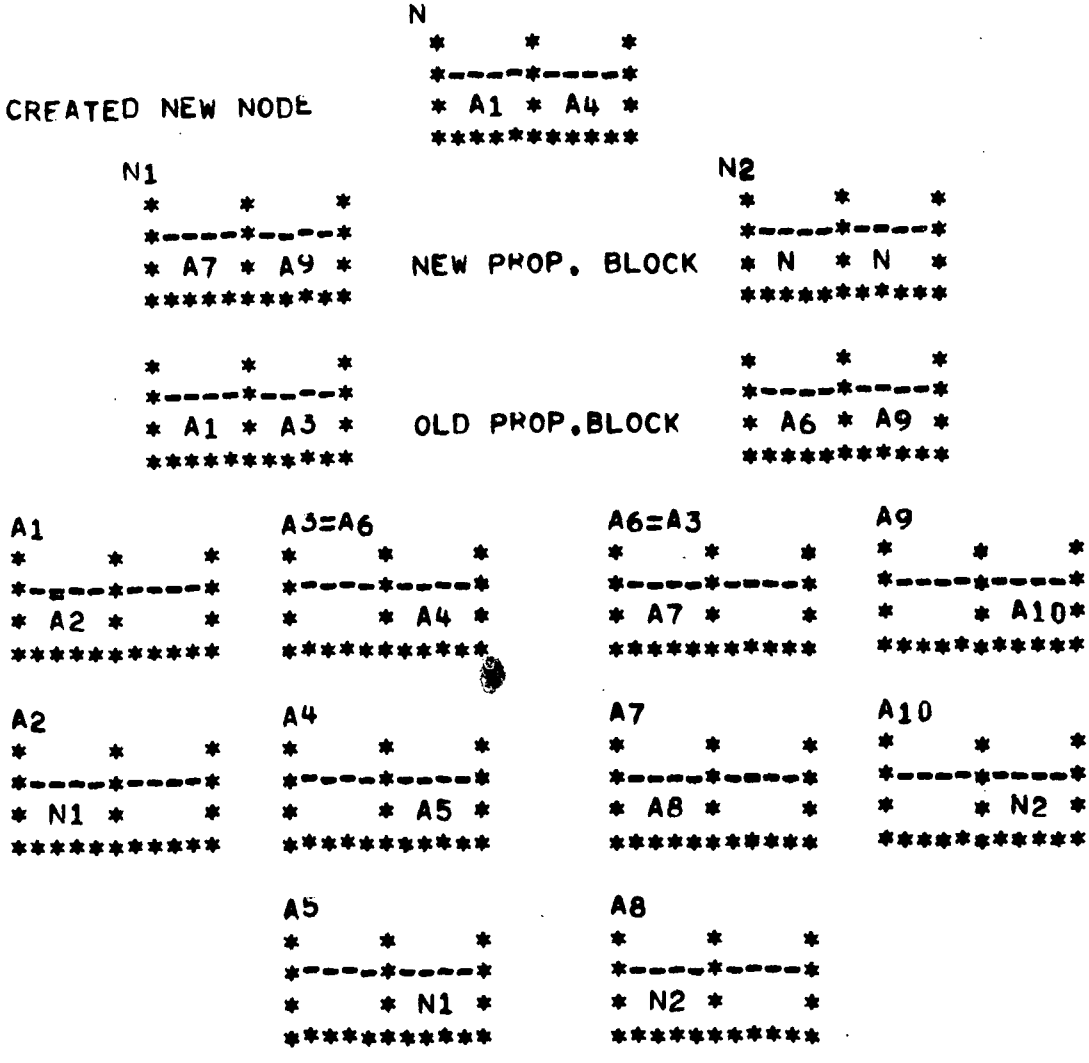
3. THE LEFT HALF OF THE SECOND WORD OF NM POINTS TO N.

A CORRESPONDING LINKAGE STRUCTURE IS DEFINED FOR PCOB OF N USING THE RIGHT FIELDS OF THE SECOND WORD OF THE NODES.

THE LINKAGE STRUCTURE IS ILLUSTRATED BELOW FOR A 2-NODE CONTRACTION:

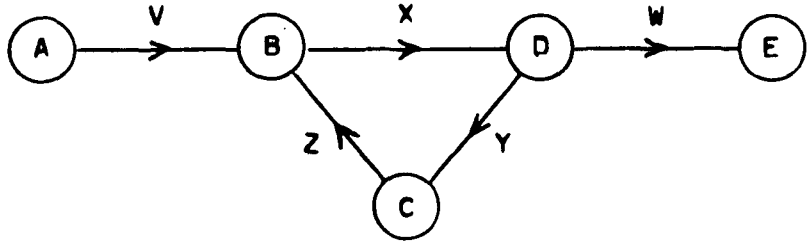


GIVES THE FOLLOWING LINKAGE (ONLY THE SECOND WORDS OF THE BLOCKS ARE SHOWN):



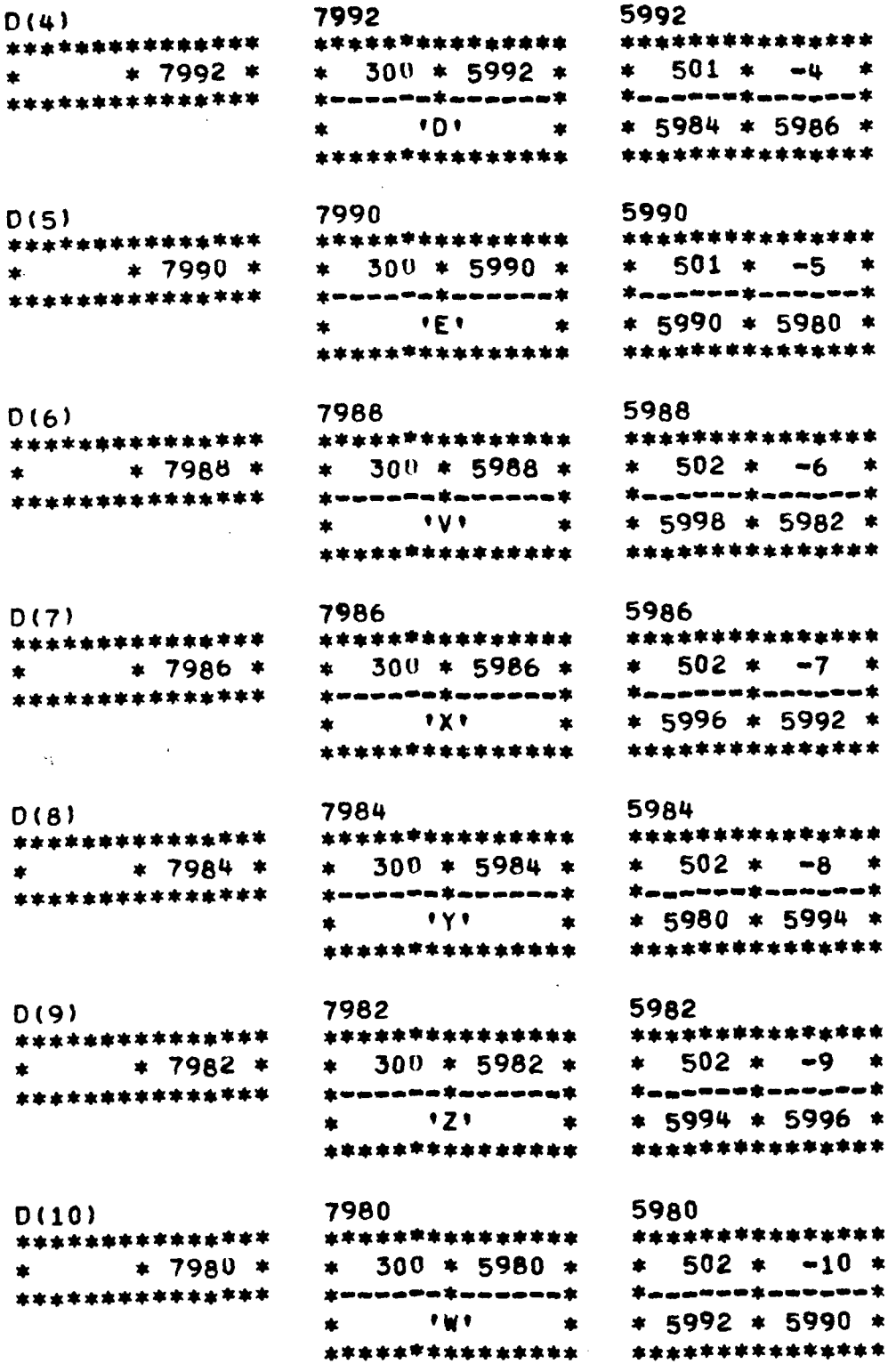
4. EXAMPLE OF A GRAPH STRUCTURE

IN THIS EXAMPLE, IT IS ASSUMED THAT 10 ELEMENTS WERE CREATED. THE ELEMENTS HAVE AN ALPHANUMERIC PROPERTY WITH '300' AS THE PROPERTY IDENTIFIER (ADDRESS OF THE PROPERTY HEADER). FURTHERMORE THE ELEMENTS WERE ASSIGNED TO A GRAPH AS NODES AND ARCS SUCH THAT THE GRAPH IDENTIFIER IS '500', THUS '501' AND '502' CORRESPONDS TO THE NODE AND ARC IDENTIFIER, RESPECTIVELY. THE GRAPH HAS THE FOLLOWING STRUCTURE:



THE FOLLOWING FIGURE GIVES THE STORAGE STRUCTURE OF THE ABOVE EXAMPLE:

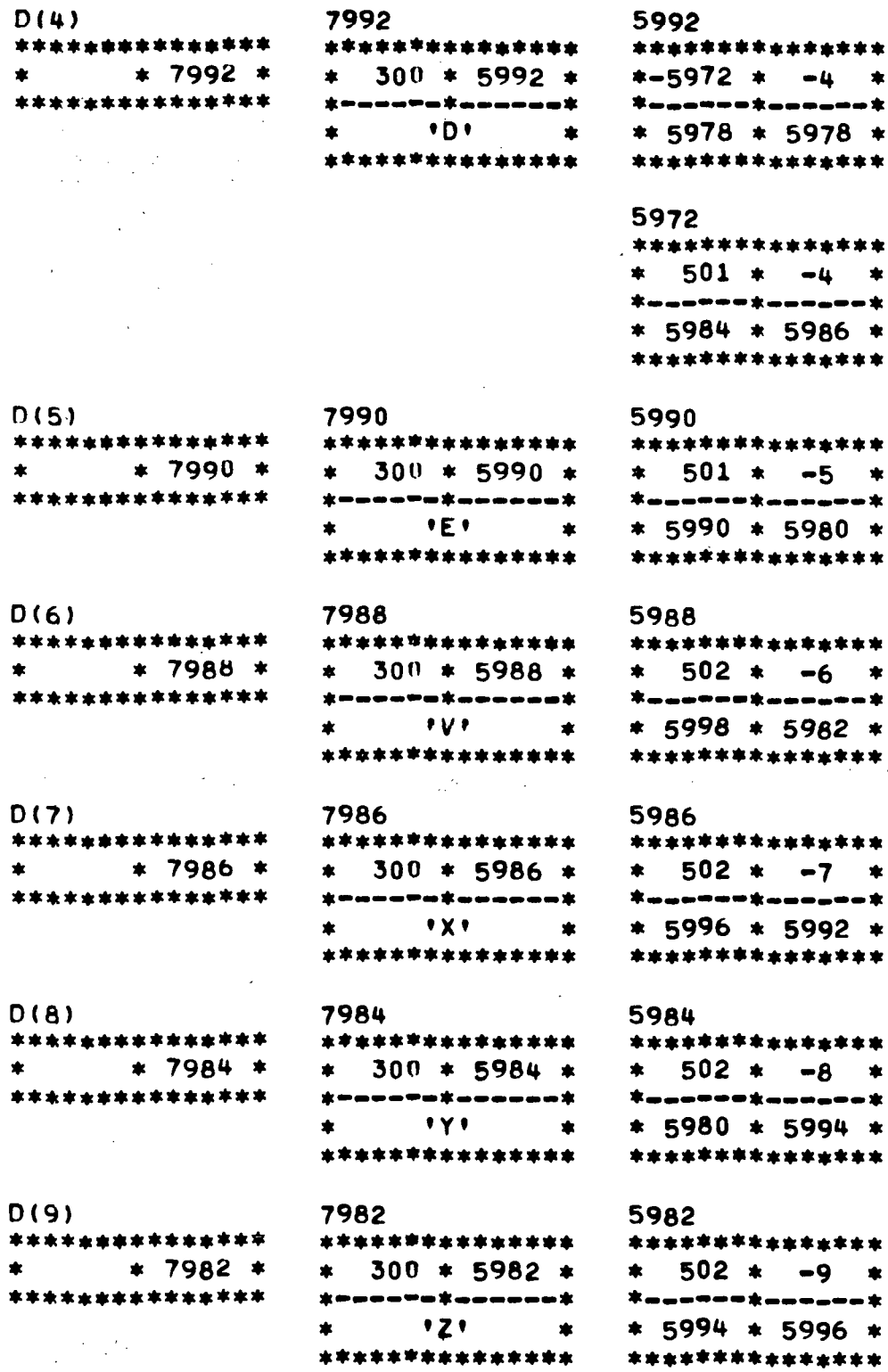
```
D(1)                7998                5998
**************      **************      **************
*      * 7998 *      *  300 * 5998 *     *  501 *  -1  *
**************      *------*------*     *------*------*
                    *     'A'     *     * 5988 * 5998 *
                    **************      **************

D(2)                7996                5996
**************      **************      **************
*      * 7996 *      *  300 * 5996 *     *  501 *  -2  *
**************      *------*------*     *------*------*
                    *     'B'     *     * 5986 * 5988 *
                    **************      **************

D(3)                7994                5994
**************      **************      **************
*      * 7994 *      *  300 * 5994 *     *  501 *  -3  *
**************      *------*------*     *------*------*
                    *     'C'     *     * 5982 * 5984 *
                    **************      **************
```

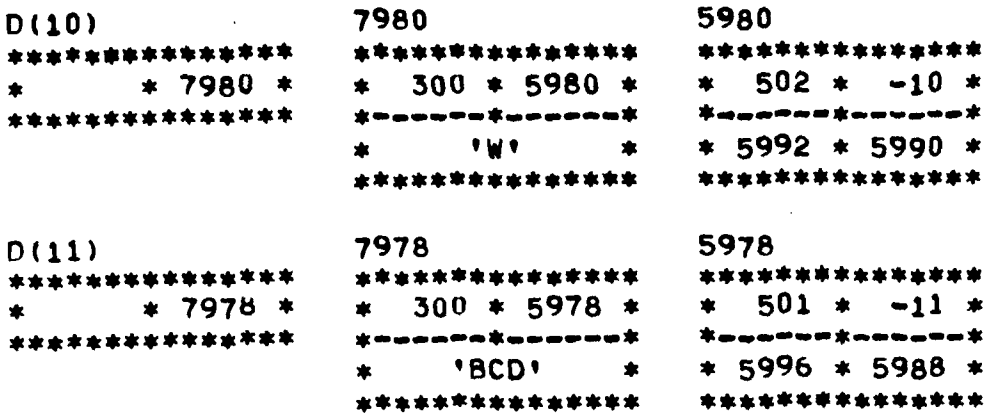

D(4)
7992
5992
7992
300 5992
501 -4
'D'
5984 5986
D(5)
7990
5990
7990
300 5990
501 -5
'E'
5990 5980
D(6)
7988
5988
7988
300 5988
502 -6
'V'
5998 5982
D(7)
7986
5986
7986
300 5986
502 -7
'X'
5996 5992
D(8)
7984
5984
7984
300 5984
502 -8
'Y'
5980 5994
D(9)
7982
5982
7982
300 5982
502 -9
'Z'
5994 5996
D(10)
7980
5980
7980
300 5980
502 -10
'W'
5992 5990

ASSUMING THE GRAPH IS CONTRACTED, SUCH THAT THE NODES B, C AND D WERE CONTRACTED INTO A NEW NODE WITH ALPHANUMERIC PROPERTY 'BCD':

THE CHANGED STORAGE STRUCTURE IS AS FOLLOWS:

```
D(1)
**************
*       * 7998 *
**************

7998
**************
*  300 * 5998 *
*-----*------*
*      'A'     *
**************

5998
**************
*  501 *  -1   *
*------*------*
* 5988 * 5998 *
**************

D(2)
**************
*       * 7996 *
**************

7996
**************
*  300 * 5996 *
*-----*------*
*      'B'     *
**************

5996
**************
*-5976 *  -2   *
*------*------*
* 5994 * 5994 *
**************

5976
**************
*  501 *  -2   *
*------*------*
* 5986 * 5982 *
**************

D(3)
**************
*       * 7994 *
**************

7994
**************
*  300 * 5994 *
*-----*------*
*      'C'     *
**************

5994
**************
*-5974 *  -3   *
*------*------*
* 5980 * 5992 *
**************

5974
**************
*  501 *  -3   *
*------*------*
* 5982 * 5984 *
**************
```


D(4)
7992
5992
* 7992 *
300 * 5992 *
-5972 * -4 *
'D'
5978 * 5978
5972
501 * -4 *
5984 * 5986 *
D(5)
7990
5990
* 7990 *
300 * 5990 *
501 * -5 *
'E'
5990 * 5980 *
D(6)
7988
5988
* 7988 *
300 * 5988 *
502 * -6 *
'V'
5998 * 5982 *
D(7)
7986
5986
* 7986 *
300 * 5986 *
502 * -7 *
'X'
5996 * 5992 *
D(8)
7984
5984
* 7984 *
300 * 5984 *
502 * -8 *
'Y'
5980 * 5994 *
D(9)
7982
5982
* 7982 *
300 * 5982 *
502 * -9 *
'Z'
5994 * 5996 *


D(10)
7980
5980
* 7980 *
* 300 * 5980 *
* 502 * -10 *
'W'
* 5992 * 5990 *
D(11)
7978
5978
* 7978 *
* 300 * 5978 *
* 501 * -11 *
'BCD'
* 5996 * 5988 *

5. THE FGRAAL COMPILER

THE FGRAAL COMPILER TRANSLATES FGRAAL SOURCE PROGRAMS INTO OBJECT CODE. TO SIMPLIFY ITS USE, IT ACCEPTS THE SAME INPUT (SYMBOLIC ELEMENTS OR CARDS), PRODUCES THE SAME OUTPUT (RELOCATABLE BINARY ELEMENTS) AND IS CALLED IN A SIMILAR WAY AS ALL LANGUAGE PROCESSORS ON THE UNIVAC 1108 UNDER EXEC 8. THE FOLLOWING SECTIONS DESCRIBE THE UNIVERSITY OF MARYLAND RALPH COMPILER, AND THE MODIFICATIONS NECESSARY TO EXTEND IT TO FGRAAL.

FGRAAL IS IMBEDDED INTO FORTRAN, AND MOST OF THE SYNTACTIC RULES OF FORTRAN APPLY ALSO TO THE EXTENSION. THE STATEMENTS MUST BE ANALYZED FAIRLY THOROUGHLY TO DETERMINE THEIR NATURE. FOR INSTANCE, THE EXPRESSION 'A(I)' CAN BE A FORTRAN FUNCTION CALL, A REFERENCE TO A SUBSCRIPTED FORTRAN VARIABLE, OR A REFERENCE TO A PROPERTY IN FGRAAL, DEPENDING ONLY ON THE WAY IN WHICH 'A' AND 'I' ARE DECLARED. ANY OF THESE CAN OCCUR IN A FORTRAN EXPRESSION. SIMILARLY, THE STATEMENT 'S=T' MUST BE TRANSLATED DIFFERENTLY WHEN 'S' AND 'T' ARE SETS OR LISTS THAN WHEN THEY ARE NORMAL FORTRAN VARIABLES.

FOR THIS REASON, AND BECAUSE THE UNIVERSITY OF MARYLAND RALPH COMPILER WAS READILY AVAILABLE AND COULD BE MODIFIED EASILY TO ACCEPT THE NEW FEATURES, WE CHOSE TO MODIFY IT FOR OUR PURPOSE.

WRITING A PREPROCESSOR INSTEAD, WHICH WOULD TRANSLATE FGRAAL INTO FORTRAN (WITH APPROPRIATE SUBROUTINE CALLS SUBSTITUTED FOR FGRAAL CONSTRUCTS) WOULD HAVE REQUIRED A VERY THOROUGH ANALYSIS OF THE PROGRAM, DUPLICATING MUCH OF THE WORK OF A COMPILER.

THE OTHER EXTREME, WRITING A COMPILER FROM SCRATCH, WOULD HAVE HAD THE ADVANTAGE OF BEING ABLE TO CONTROL THE METHODS OF COMPILATION MORE COMPLETELY, BUT THE AMOUNT OF WORK INVOLVED WOULD HAVE BEEN GREAT, AND MUCH OF IT WOULD HAVE BEEN A DUPLICATION OF FUNCTIONS OF ALREADY EXISTING COMPILERS.

5.1. THE RALPH COMPILER

THE FOLLOWING IS A BRIEF DESCRIPTION OF RALPH, WHICH CANNOT DO FULL JUSTICE TO ALL ITS FEATURES.

RALPH (REENTRANT ALGORITHMIC LANGUAGE PROCESSOR) IS A FOUR PASS COMPILER, DESIGNED TO COMPILE PROGRAMS QUICKLY, WITH LESS EXTENSIVE OPTIMIZATION THAN THE UNIVAC FORTRAN V COMPILER.

PASS ONE READS THE SOURCE PROGRAM AND TRANSLATES IT TO INTERMEDIATE OBJECT CODE (TRIPLES). AT THE SAME TIME IT BUILDS THE SYMBOL TABLES. THE STATEMENT RECOGNIZER IS TABLE DRIVEN. THE VARIOUS STATEMENTS ARE PROCESSED IN INDIVIDUAL PARTS OF THE COMPILER, WHICH USE A COMMON POOL OF SUBROUTINES. THESE INCLUDE A SYNTAX SCANNER, AN EXPRESSION COMPILER, THE SYMBOL TABLE ROUTINES, AND A ROUTINE FOR OUTPUTTING TRIPLES TO A SCRATCH FILE, WHICH DOES

ALL OPTIMIZATION.

PASS TWO CLEANS UP THE SYMBOL TABLE, PROCESSES EQUIVALENCE STATEMENTS, AND ALLOCATES DATA STORAGE.

PASS THREE CONVERTS THE INTERMEDIATE OBJECT CODE TO MACHINE INSTRUCTIONS, SOME OF WHICH MAY NOT YET BE COMPLETE. IT READS THE TRIPLES, DETERMINES THE MODES OF THE OPERANDS, AND FINDS A TABLE ENTRY FOR THAT PARTICULAR COMBINATION. FROM THAT IT DETERMINES THE MODE OF THE RESULT, IF ANY, AND FINDS THE DEFINITION SEQUENCE FOR THE TRIPLE. THE DEFINITION SEQUENCES CONTAIN PSEUDO INSTRUCTIONS (FOR REGISTER ALLOCATION, TRANSFER OF CONTROL, DEFINITION OF REGISTER CONTENTS, ETC.) AS WELL AS MACHINE INSTRUCTIONS, WHOSE REGISTER AND OPERAND FIELDS ARE FILLED WITH SPECIAL CODES. AN INTERPRETER ANALYZES THESE, REPLACES THE FIELDS BY REAL ADDRESSES, AND WRITES THESE, WITH ANY RELOCATION INFORMATION, TO A SCRATCH FILE. INCOMPLETE ADDRESSES (FORWARD REFERENCES TO LABELS, ETC.) ARE FLAGGED.

PASS FOUR READS THE OUTPUT FROM PASS THREE, MODIFIES THE INCOMPLETE INSTRUCTIONS, AND WRITES THE FINISHED RELOCATABLE ELEMENT. IT ALSO PRINTS AN OBJECT CODE LISTING WHEN REQUESTED.

5.2. MODIFICATIONS TO RALPH

THESE FALL INTO TWO GROUPS: SOME ARE LANGUAGE RELATED, AND WOULD HAVE TO BE TREATED SIMILARLY IN MOST IMPLEMENTATIONS, OTHERS ARE DIRECTLY RELATED TO THE COMPILER WHICH IS BEING MODIFIED.

5.2.1. DATA TYPES

THERE ARE SEVERAL NEW VARIABLE TYPES: SETS, STAQUES, PROPERTIES, AND GRAPHS. SETS, STAQUES, AND PROPERTIES OCCUPY SINGLE WORDS IN THE PROGRAMS, WHICH MAY POINT INTO DYNAMIC STORAGE AT EXECUTION TIME. GRAPHS CONSIST OF THREE WORDS, ONE HAS INFORMATION ABOUT THE GRAPH, THE OTHER TWO ABOUT THE ARCS AND NODES, RESPECTIVELY. SET IS A VARIABLE TYPE, LIKE INTEGER OR REAL. STAQUES AND PROPERTIES HAVE ONE OF THE TYPES INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, OR SET ASSOCIATED WITH THEM.

MOST OF THESE NEW VARIABLES REQUIRE THAT AN INITIAL VALUE IS COMPILED FOR THEM. THIS IS DONE AT THE END OF PASS THREE, USING THE INFORMATION IN THE SYMBOL TABLE.

5.2.2. IMPLEMENTATION OF NEW STATEMENTS

DECLARATION STATEMENTS ARE HANDLED IN THE SAME WAY AS FORTRAN DECLARATIONS. THE SYMBOL IS ENTERED INTO THE MODE TABLE, AND, IF NECESSARY, INTO THE DIMENSION TABLE.

ASSIGN, DETACH, AND REMOVE STATEMENTS ARE TRANSLATED INTO CALLS TO APPROPRIATE SUBROUTINES.

THE SAVE AND RESET STATEMENTS ARE NOT YET IMPLEMENTED.

THE TWO NEW LOOP STATEMENTS ('WHILE' AND 'FOR ALL') CAN BE HANDLED SIMILAR TO THE DO LOOP. MOST OF THE CODE IS GENERATED AT THE BEGINNING OF THE LOOPS, AND ONLY THE LOOP VARIABLE HAS TO BE REMEMBERED FOR THE TERMINATION, AND A TRANSFER AND A LABEL HAVE TO BE GENERATED AT THE END. ONE SPECIAL PROBLEM CONCERNING 'FOR ALL' LOOPS IS DISCUSSED IN SECTION 5.3.4.

5.2.3. IMPLEMENTATION OF NEW SYNTACTIC FEATURES

THE SET OPERATIONS ARE SIMPLY ENTERED INTO A TABLE OF LEGAL OPERATORS. THEY ALL PRODUCE SUBROUTINE CALLS.

THE GRAPH OPERATIONS (BOUNDARY OPERATIONS) ARE HANDLED SIMILARLY, EXCEPT THAT THEY ALL CALL THE SAME SUBROUTINE, AND THE SPECIFIC OPERATION IS TRANSMITTED THROUGH A FLAG WORD.

THE FUNCTIONS 'NODES', 'ARCS', AND 'DOMAIN' ARE TRANSLATED INTO CONSTANT OFFSETS FROM THE GRAPH OR PROPERTY, AND THEIR RESULTS ARE SETS. THIS CAUSED SOME DIFFICULTY BECAUSE OF THE WAY IN WHICH RALPH PASSES CONSTANT OFFSETS FROM PASS ONE TO PASS THREE.

PROPERTIES CAN APPEAR ON EITHER SIDE OF ASSIGNMENT STATEMENTS. THE TWO CASES HAVE A DIFFERENT MEANING, AND EACH MUST BE TRANSLATED DIFFERENTLY.

THE SUBSET FUNCTION PRESENTED A PARTICULARLY DIFFICULT PROBLEM. IF IT IS NOT TO BE TRANSLATED INLINE, WHICH WOULD GENERATE TOO MUCH CODE, IT REQUIRES A CALL BY NAME TO EVALUATE THE LOGICAL EXPRESSION. THIS IS ACHIEVED BY TRANSLATING THE LOGICAL EXPRESSION INTO AN INTERNAL FUNCTION, AND PASSING THE ADDRESS OF THAT FUNCTION TO THE SUBSET ROUTINE.

THE ARGUMENTS TO LIST ASSIGNMENT STATEMENTS ARE PASSED IN ONE ARRAY, CONTAINING THE ADDRESS OF EACH ITEM, AND A FLAG INDICATING WHETHER OR NOT THE VARIABLE IS ANOTHER LIST, AND WHETHER OR NOT IT NEEDS TO BE COPIED (FOR LISTS AND SETS).

5.2.4. HANDLING OF TEMPORARIES

TEMPORARY RESULTS FROM SET OPERATIONS PRESENT A NEW PROBLEM: THEY DO NOT CONTAIN A VALUE WHICH CAN BE SIMPLY FORGOTTEN, BUT THEY POINT TO AN ALLOCATED AREA IN DYNAMIC STORAGE WHICH HAS TO BE FREED. THIS OCCURS IN THE EVALUATION OF EXPRESSIONS, WHEN PASSING SET EXPRESSIONS TO A SUBROUTINE AS AN ARGUMENT, AND DURING THE EXECUTION OF 'FOR ALL' LOOPS, FOR WHICH A SET HAS TO BE KEPT THROUGHOUT THE EXECUTION OF THE LOOP.

FREEING EACH TEMPORARY EXPLICITLY WOULD CREATE MUCH EXTRA CODE, SO THAT A BETTER METHOD HAD TO BE FOUND. SINCE ALL SET OPERATIONS ARE EXECUTED IN SUBROUTINES, A FLAG IS PASSED IN THE CALLING SEQUENCE, INDICATING WHEN A TEMPORARY SET RESULT IS NO LONGER NEEDED. THE SUBROUTINE CAN THEN USE THE SET WITHOUT COPYING IT, IF IT WISHES TO DO SO. OTHERWISE IT HAS TO FREE THE SET BEFORE IT EXITS. THIS IS MADE THE RESPONSIBILITY OF THE SUBROUTINE. SET ARGUMENTS TO FORTRAN SUBROUTINES CAN BE HANDLED IN THE SAME WAY, PROVIDED THAT THE SUBROUTINE EXIT SEQUENCE TAKES CARE OF

FREEING ALL THE SETS.

TRANSFERS OUT OF 'FOR ALL' LOOPS ARE MORE DIFFICULT. SINCE THE TRANSFER MAY BE IMBEDDED INTO A CALLING SEQUENCE, IT IS NOT POSSIBLE TO SIMPLY REDEFINE THE JUMP INSTRUCTION. INSTEAD, FOR EACH TRANSFER OUT OF A 'FOR ALL' LOOP, A DUMMY LABEL IS CREATED AT THE END OF THAT LOOP, FOLLOWED BY A RELEASE OF THE SET, AND A JUMP TO THE REAL LABEL, OR A JUMP TO A DUMMY LABEL AT THE END OF THE NEXT 'FOR ALL' LOOP.

5.3. OPTIMIZATION

REPEATED SET OPERATIONS ARE ELIMINATED IN THE SAME WAY AS ALL REPEATED ARITHMETIC OPERATIONS, AS LONG AS NO LABELS INTERVENE. THE SAME HOLDS FOR PROPERTY RETRIEVALS AND THE NONDESTRUCTIVE LIST RETRIEVALS. THIS FEATURE CAN BE TURNED OFF BY USING THE 'O' OPTION ON THE COMPILER CALL CARD.

OPERATIONS OF THE FORM

SET = SET .UN. X OR SET = SET .DF. X

ARE FREQUENT ENOUGH THAT IT SEEMED WORTHWHILE TO HAVE SPECIAL ENTRIES FOR THEM. THIS REDUCES THE LENGTH OF THE CALLING SEQUENCE BY ONE HALF, AND MAY ALSO BE USED BY THE SUBROUTINE TO AVOID UNNECCESSARY COPYING OF THE SET.

6. COMPILER GENERATED INSTRUCTION SEQUENCES

THIS CHAPTER CONTAINS THE DESCRIPTION OF THE INSTRUCTION SEQUENCES GENERATED BY THE FGRAAL COMPILER FOR THE SPECIAL FEATURES OF FGRAAL. THE FGRAAL COMPILER IS AN EXTENSION OF THE RALPH COMPILER, THUS IT GENERATES INSTRUCTION FOR THE FORTRAN STATEMENTS ALSO. THERE IS NO ATTEMPT MADE TO DESCRIBE THESE INSTRUCTION SEQUENCES, ONLY THE SEQUENCES REVELANT TO FGRAAL FEATURES ARE DESCRIBED HERE.

6.1. DECLARATION STATEMENTS

DECLARED SETS, PROPERTIES, LISTS AND GRAPHS RECEIVE INITIAL DATA ASSIGNMENTS BY THE COMPILER. THESE DATA ARE INITIALIZED AS FOLLOWS:

SET		0	0
PROPERTY		-0	TYPE
LIST		-0	TYPE
GRAPH	G	-0	XXXX YYYY 3
	G+1	-0	1
	G+2	-0	2

WHERE XXXX = 0 FOR NODE GRAPH
= 1 FOR NODE/ARC GRAPH
YYYY = 0 FOR UNDIRECTED GRAPH
= 1 FOR DIRECTED GRAPH
TYPE = 0 FOR SET TYPE
= 4 FOR LOGICAL TYPE
= 5 FOR INTEGER TYPE
= 6 FOR REAL TYPE
= 7 FOR DOUBLE PRECISION TYPE
= 8 FOR COMPLEX TYPE

6.2. CALLING SEQUENCES

THE COMPILER GENERATES CALLING SEQUENCES FOR THE FGRAAL LIBRARY ROUTINES. THESE CALLING SEQUENCES ARE IN THE SAME FORM AS THE FORTRAN CALLING SEQUENCES, I.E.

```
LMJ     X11,LIBR
 +      ARG1
 ..........
 +      ARGN
 +      W.B.
```

WHERE 'LIBR' IS THE NAME OF A LIBRARY PROGRAM, ARG1,...,ARGN ARE THE ADDRESSES OF THE ARGUMENTS, W.B. IS THE WALK BACK WORD. REGISTERS A0-A5 ARE ASSUMED TO BE FREE TO USE BY THE LIBRARY PROGRAMS WITHOUT SAVING THEM. RESULTS ARE RETURNED IN A_0 OR IN A_0-A_1.

SETS IN THE CALLING SEQUENCES HAVE AN ADDITIONAL FEATURE. BESIDES THE ADDRESS OF THEIR SET HEADER, THERE IS A FLAG IN BIT POSITION 12:

```
**************************************
*            F     * ADDRESS OF SET   *
**************************************
```

WHERE THE FLAG IS SET BY THE COMPILER TO ONE, IF THE SET IS A TEMPORARY SET SUCH THAT IT SHOULD BE CLEARED AFTER IT IS USED AS AN INPUT TO THE LIBRARY PROGRAM. OTHERWISE IT IS SET TO ZERO. IN ALL ARGUMENTS THE X FIELD (INDEX REGISTER MODIFICATION) MAY BE NONZERO, AND INDIRECT ADDRESSING MAY ALSO BE SPECIFIED.

THE FLAG POSITION IS ALSO USED IN LIST ASSIGNMENTS FOR SIMILAR REASONS BUT IT IS EXTENDED FOR IDENTIFYING LISTS.

6.3. FREEING A SET OR LIST

THE COMPILER RECOGNIZES THE SIMPLE ASSIGN STATEMENTS,

```
 S = .EMPTY.          AND          L = .NIL.
(S = & )                         (L = # )
```

AND TRANSLATES THEM INTO THE FOLLOWING CALLING SEQUENCES, RESPECTIVELY:

```
LMJ     X11,G$SZRO              LMJ     X11,G$LZRO
 +      LOC(S)                   +      LOC(L)
 +      W.B.                     +      W.B.
```

6.4. SET - OPERATIONS, -RELATIONS, -ASSIGNMENT

LET S AND T BE SETS, THE FOLLOWING SET OPERATIONS :

```
S .UN. T          S .DF. T
S .IT. T          S .SM. T
```

AND SET RELATIONS :

```
S .EQ. T          S .NE. T
S .IN. T          S .NIN. T
```

ARE TRANSLATED INTO THE CALLING SEQUENCES OF THE FORM :

```
LMJ     X11,G$XXXX
 +      F,S
 +      F,T
 +      W.B.
```

WHERE G$XXXX IS AS FOLLOWS:

```
G$SUN   FOR .UN.
G$SIT   FOR .IT.
G$SDF   FOR .DF.
G$SSM   FOR .SM.
G$SIN   FOR .IN.
G$SNIN  FOR .NIN.
G$SNE   FOR .NE. AND .EQ.
```

THE SET HEADER FOR THE RESULT OF THE ABOVE SET OPERATIONS WILL BE RETURNED IN REGISTER A0. THE LOGICAL RESULT OF THE SET RELATIONAL OPERATION WILL ALSO BE RETURNED IN REGISTER A0.

6.5. SPECIAL SET FUNCTIONS

6.5.1. CREATE FUNCTION

THE FUNCTION

```
CREATE(0)
```

IS TRANSLATED INTO THE CALLING SEQUENCE

```
LMJ     X11,G$CRT0
 +      W.B.
```

AND THE LIBRARY ROUTINE RETURNS THE ATOMIC SET IN A0.
THE CONDITIONAL CREATE FUNCTION,

CREATE(P1,V1,P2,V2,...,PN,VN)

WITH PROPERTY IDENTIFIERS PI AND PROPERTY VALUES VI ARE TRANSLATED INTO THE CALLING SEQUENCE

```
LMJ     X11,G$CRT
 +      A
 +      N
 +      W.B.
```

WHERE N IS THE NUMBER OF PAIRS OF PROPERTY IDENTIFIERS AND VALUES. A IS THE ADDRESS OF ONE ARRAY CONTAINING THE ADDRESSES OF ALL PROPERTY HEADERS FOLLOWED BY ALL PROPERTY VALUES:

```
A    +   P1              A+N   +   V1
     +   P2                    +   V2
     . . .                     . . .
     +   PN                    +   VN
```

THESE ARRAYS ARE GENERATED BY THE COMPILER. WHEN A PROPERTY VALUE IS OF TYPE SET, THEN THE ADDRESS OF THE VALUE, VI, IS FLAGGED. IF THE FLAG IS ONE THEN THE SET WILL BE FREED BY THE LIBRARY ROUTINE BEFORE RETURN.

6.5.2. SUBSET FUNCTION

THE EXPRESSION

SUBSET (X,<LOGICAL EXP.>)

IS TRANSLATED BY THE COMPILER AS

```
LMJ     X11,G$SUBS
 +       0
 +       X
 J      'FUNCTION'
 +      (W. B.)
```

WHERE 'FUNCTION' IS THE ADDRESS OF AN INTERNAL SUBPROGRAM WHICH EVALUATES THE LOGICAL EXPRESSION.

THE FUNCTION

SUBSET (S, X, <LOGICAL EXP.>)

IS TRANSLATED SIMILARLY, BUT THE FIRST ARGUMENT IS THE ADDRESS OF

THE SET S.

6.5.3. ELT, INDEX, SIZE AND PARITY FUNCTIONS

THE FUNCTIONS,

ELT(I,S) , INDEX(X,S) , SIZE(S) , PARITY(S) ,

ARE TRANSLATED AS STANDARD FORTRAN FUNCTIONS, BUT THEIR NAMES ARE REPLACED BY GELT, GINDX, G$SIZE, AND G$PARI.

6.6. PROPERTY ASSIGNMENT AND RETRIEVAL

THE RETRIEVAL OF A PROPERTY VALUE,

P(X)

OF AN ATOMIC SET X IS TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

```
LMJ     X11,G$PRRT
 +      P
 +      X
 +      W.B.
```

WHERE P AND X ARE ADDRESSES OF THE PROPERTY HEADER AND SET HEADER, RESPECTIVELY. THE VALUE IS RETURNED IN A0 (OR $A0-A_1$ FOR DOUBLE PRECISION AND COMPLEX) BY THE LIBRARY ROUTINE.

THE ASSIGNMENT OF A PROPERTY VALUE,

P(X) = V

TO AN ATOMIC SET X IS TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

```
LMJ     X11,G$PRST
 +      P
 +      V
 +      X
 +      W.B.
```

WHERE P AND X ARE AS BEFORE, AND V IS THE ADDRESS WHERE THE VALUE IS FOUND. IN CASE OF A SET-VALUE, V IS FLAGGED. IF THE FLAG IS ZERO, THEN THE SET IS COPIED, IF IT IS ONE, THE SET IS LINKED INTO THE PROPERTY CHAIN DIRECTLY.

6.7. LIST ASSIGNMENT AND LIST FUNCTIONS

6.7.1. LIST ASSIGNMENT STATEMENT

THE STATEMENT,

```
L = #    OR    L = .NIL.
```

IS TRANSLATED INTO THE CALLING SEQUENCE

```
LMJ     X11,G$LZRO
 +      L
 +      W.B.
```

THE STATEMENT,

```
L = A1 : A2 : ... : AN
```

IS TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

```
LMJ     X11,G$LIST
 +      ARG
 +      N
 +      L
 +      W.B.
```

WHERE N IS THE NUMBER OF ARGUMENTS, L IS THE ADDRESS OF THE LIST HEADER AND ARG IS THE ADDRESS OF A COMPILER GENERATED ARRAY CONTAINING THE ADDRESSES OF THE N ARGUMENTS INVOLVED IN THE CONCATENATION.

```
ARG    +    F1,LOC(A1)
       +    F2,LOC(A2)
       . . . .
       +    FN,LOC(AN)
```

THESE ADDRESSES ARE FLAGGED IN BIT POSITIONS 12 AND 13 WITH THE FOLLOWING CODE:

```
FI = 0  AI IS A VARIABLE WHICH HAS TO BE COPIED
   = 1  AI IS A LIST WHICH HAS TO BE COPIED
   = 2  AI IS A SET WHICH NEED NOT BE COPIED
   = 3  AI IS A LIST WHICH CAN BE CONCATENATED WITHOUT
        COPYING IT
```

6.7.2. LIST FUNCTIONS

THE FUNCTIONS,

```
FIRST(L) , DFIRST(L) ,
LAST(L)  , DLAST(L)
```

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

```
LMJ     X11,G$LGET
 +      T
 +      L
 +      W.B.
```

WHERE

```
T  = 0   FOR FIRST
   = 1   FOR DFIRST
   = 2   FOR LAST
   = 3   FOR LAST AND DLAST
```

AND L IS THE ADDRESS OF THE LIST HEADER.
THE VALUE IS RETURNED IN A0 (A0-A1 FOR DBL.PR. AND COMPLEX) BY THE LIBRARY ROUTINE.

6.8. GRAPH OPERATIONS

6.8.1. ASSIGN STATEMENT

THE STATEMENTS,

```
ASSIGN G, X
ASSIGN G, X - Y
ASSIGN G, X - Y, Z
```

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCES, RESPECTIVELY:

```
LMJ     X11,G$ASG1
 +      G
 +      X
 +      W.B.

LMJ     X11,G$ASG2
 +      G
 +      X
 +      Y
 +      W.B.

LMJ     X11,G$ASG3
 +      G
 +      X
```

```
+       Y
+       Z
+       W.B.
```

WHERE G IS THE ADDRESS OF THE GRAPH HEADER, X, Y, Z ARE THE ADDRESSES OF THE ATOMIC SETS.

6.8.2. DETACH STATEMENT

THE STATEMENTS,

```
DETACH G
DETACH G, S
DETACH G, S - T
```

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCES, RESPECTIVELY:

```
LMJ     X11,G$DET1
 +      G
 +      W.B.

LMJ     X11,G$DET2
 +      G
 +      S
 +      W.B.

LMJ     X11,G$DET3
 +      G
 +      S
 +      T
 +      W.B.
```

WHERE G IS THE ADDRESS OF THE GRAPH HEADER, S AND T ARE THE ADDRESSES OF THE CORRESPONDING SET HEADERS.

6.8.3. GRAPH FUNCTIONS

THE GRAPH FUNCTIONS,

STAR(G,S), COB(G,S), PBD(G,S), ETC.

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

```
LMJ     X11,G$BDRY
 +      T
 +      G
 +      S
 +      W.B.
```

WHERE G IS THE ADDRESS OF THE GRAPH HEADER, S IS THE ADDRESS OF

THE SET HEADER, AND T IS DEFINED AS FOLLOWS (OCTAL):

	T
STAR	202
PSTAR	102
NSTAR	002
COB	212
PCOB	112
NCOB	012
ADJ	200
PADJ	100
NADJ	000
INC	201
PINC	001
NINC	101
BD	211
PBD	011
NBD	111

6.9. ITERATIVE STATEMENTS

6.9.1. WHILE STATEMENT

THE STATEMENT

```
DO  #  WHILE  <LOG.EXPR,>
```

IS TRANSLATED AS

```
F$XXXA  ( TRANSLATION
            OF
          LOGICAL
            EXPRESSION )
        JZ    A0,F$XXXB
           .........
 #         .........
        J     F$XXXA
F$XXXB     .........
```

6.9.2. FOR ALL STATEMENT

THE STATEMENT

```
DO  #  FOR ALL  X .IN. S
```

IS TRANSLATED AS

```
        LMJ    X11,G$D01
```

```
                +       S
                +       TEMP
                +       W.B.
F$XXXA          LMJ     X11,G$DO2
                +       X
                J       F$XXXB
                +       TEMP
                +       W.B.

                 .........

                 .........
                J       F$XXXA
F$XXXB          LMJ     X11,G$DO3
                +       TEMP
                +       W.B.
```

WHERE 'TEMP' IS A TEMPORARY LOCATION ASSIGNED BY THE COMPILER. THE THREE LIBRARY ROUTINES, G$DO1-3, ARE FOR INITIALIZATION, ITERATION AND TERMINATION OF THE LOOP, RESPECTIVELY. THE COMMUNICATION BETWEEN THESE ROUTINES IS ACCOMPLISHED BY THE CONTENTS OF 'TEMP'.

THE LAST ROUTINE, G$DO3, IS ALSO CALLED IF THE DO RANGE IS LEFT BY TRANSFER STATEMENTS (ABNORMAL EXIT).

6.10. REMOVE STATEMENTS

THE STATEMENT

```
REMOVE  P,...,Q(S),...T,...
```

WITH PROPERTIES P,Q,..., SETS S,T,... ARE TRANSLATED INTO CONSECUTIVE SUBROUTINE CALLS, ONE FOR EACH ARGUMENT IN THE LIST OF THE STATEMENT.

IF THE ARGUMENT IN THE STATEMENT IS A PROPERTY, THEN THE CALLING SEQUENCE IS

```
LMJ     X11,G$RMV1
+       P
+       W.B.
```

IF THE ARGUMENT IS A PROPERTY ON A SET, THEN

```
LMJ     X11,G$RMV2
+       Q
+       S
+       W.B.
```

IF THE ARGUMENT IS A SET, THEN

```
        LMJ     X11,G$RMV3
        +       T
        +       W.B.
```

7. LIBRARY PROGRAMS

THIS CHAPTER DESCRIBES THE OBJECT TIME LIBRARY PACKAGE OF FGRAAL. THE ENTRY POINTS ARE SUCH THAT THE FIRST TWO CHARACTERS OF THEIR NAMES ARE 'G$'. EXCLUDED FROM THIS CONVENTION ARE THE SPECIAL FUNCTIONS OF FGRAAL SUCH AS EXPAND, CONTR, ETC.

THE LIBRARY PACKAGE IS DIVIDED FUNCTIONALLY INTO TEN GROUPS. THE FIRST FOUR OF THESE GROUPS, DATA REFERENCES IN THE DYNAMIC STORAGE, PROCEDURES, DYNAMIC STORAGE ROUTINE AND THE SET ROUTINES, ARE BASIC IN THE PACKAGE IN THE SENSE THAT THEY ARE CALLED BY THE ROUTINES IN THE OTHER PACKAGE. THE OTHER GROUPS ARE INDEPENDENT FROM EACH OTHER. THE FOLLOWING TABLE SUMMARIZES THE ELEMENTS IN THE TEN GROUPS.

ELT.	ENTRIES	TITLE
		COMMON DATA
G$DATA	G$NU0	ZERO AS LOWER LIMIT OF UNIV.SEQ.
	G$NU	LAST CREATED ELEMENT
	G$NF	FIRST FREE BLOCK
	G$NL	LOWEST BLOCK ALLOCATED
	G$DA	ADDRESS OF DYNAMIC STORAGE
	G$D	DYNAMIC STORAGE
	G$MASK	MASKING BITS FOR COLUMN SETS
		PROCEDURES
PROCS	IDYST3	FREE A 2-WORD BLOCK
	ELEMNT	OBTAIN ELEMENT FROM ITS PROP.BLOCK
	REMOV	UNCHAIN A 2-WORD BLOCK
	REMPR	REMOVE A PROPERTY BLOCK
	INSRT	INSERT A 2-WORD BLOCK INTO A CHAIN
	SAVEX	SAVE RETURN ADDRESS
	PTYP	GETS TYPE GROUP OF PROPERTY
	SETONE	SET A BIT IN A COLUMN
	SETZRO	RESET A BIT IN A COLUMN
	PTAG	SET PTAG ON A COLUMN SET
		DYNAMIC STORAGE ROUTINES
G$DYN	G$DYN	OBTAIN A 2-WORD BLOCK

		SET ROUTINES
G$SZRO	G$SZRO	FREE A SET
	G$SCZR	FREES A SET IF FLAGGED
	CFREE	FREE A SET IN COLUMN FORM
	BFREE	FREE A SET IN BLOCK FORM
	CRTRN	FREE A COLUMN INDEX
G$SASG	G$SASG	SET ASSIGNMENT
G$SOPR	G$SUN	SET UNION
	G$SSM	SYMMETRIC SUM
	G$SDF	SET DIFFERENCE
	G$SIT	SET INTERSECTION
	G$SCT	SET CONTAINMENT
	G$SEQ	SET EQUIVALENCE
	G$SNE	SET NOT EQUAL
G$SCPY	G$SCPY	COPY A SET INTO A COLUMN SET
	G$SCP1	COPY A SET INTO A CERTAIN COLUMN
G$CDIS	G$CDIS	PUT A COLUMN SET INTO BLOCK FORM
G$CGET	G$CGET	OBTAIN A FREE COLUMN
G$CZRO	G$CZRO	CLEAR A COLUMN
G$SETI	G$SETI	INITIALIZE FOR GET-NEXT ELT.
G$INI	G$INI	REGISTER INIT. FOR GET NEXT
	G$INIQ	INITIALIZATION FOR INTERNAL USE
	IN	DATA WORD FOR GET NEXT
	G$INIC	REGISTER SET-UP FOR COL.SEARCH
G$NXT	G$NXT	GET NEXT ELEMENT OF SET
	G$NXTQ	FOR INTERNAL USE
	G$NXTB	GET NEXT ELT. OF BLOCK SET
G$GNHD	G$GNH1	GENERATE HEADER FOR COLUMN SET, PREVIOUS HEADER EXISTED
	G$GNH2	PREVIOUS HEADER EXISTED BUT NEW LIMITS ARE GIVEN
	G$GNH3	NO PREVIOUS HEADER
		LIST ROUTINES
G$LZRO	G$LZRO	EMPTY A LIST
G$LGET	G$LGET	LIST FUNCTIONS
G$LIST	G$LIST	LIST ASSIGNMENT

		PROPERTY ROUTINES
G$RMV1	G$RMV1	REMOVE PROPERTY P
G$RMV2	G$RMV2	REMOVE PROPERTY ON A SET
G$RMV3	G$RMV3	REMOVE ELEMENTS FROM UNIVERSE
G$PROP	G$PRST	PROPERTY ASSIGNMENT
	G$PRRT	PROPERTY RETRIEVAL
G$PREM	G$PREM	REMOVE A PROPERTY FROM AN ELEMENT
G$PADD	G$PADD	ADD ELT. TO PROPERTY SET
G$PSUB	G$PSUB	REMOVE ELT. FROM PROPERTY SET
G$FNDF	G$FNDF	FIND FIRST ELT. WITH PROPERTY
	G$FNDL	FIND LAST ELT. WITH PROPERTY
G$GPR1	G$GPR1	FIND PROP.BLOCK OF ELT.
		GRAPH ROUTINES
G$ASG	G$ASG1	ASSIGN G, X
	G$ASG2	ASSIGN G, X-Y
	G$ASG3	ASSIGN G. X-Y, Z
G$DET1	G$DET1	DETACH FULL GRAPH
G$DET2	G$DET2	DETACH ELEMENTS FROM GRAPH
G$DET3	G$DET3	DETCH LINKS FROM GRAPH
G$BDRY	G$BDRY	GRAPH OPERATORS
G$DOMN	G$DOMN	DOMAIN OF A PROPERTY (NODES,ARCS)
G$DETA	G$DETA	DETACH AN ARC
G$ASGN	G$ASGN	ASSIGN A NODE
G$GPR2	G$GPR2	FIND GRAPH PROPERTY BLOCK
		SPECIAL FUNCTIONS
G$CRT	G$CRT	CONDITIONAL CREATE
G$CRT0	G$CRT0	CREATE ELEMENT
G$DO	G$DO1	INITIALIZATION FOR FORALL ST.
	G$DO2	RETRIEVE ELEMENT FOR FORALL ST.
	G$DO3	TERMINATE FORALL STATEMENT
G$CHCK	G$CHCK	PROPERTY CHECK
G$ELT	G$ELT	ELT FUNCTION
G$INDX	G$INDX	INDEX FUNCTION
G$SIZE	G$SIZE	SIZE FUNCTION
	G$PARI	PARITY FUNCTION
G$SUBS	G$SUBS	SUBSET FUNCTION
		DIAGNOSTIC ROUTINES
G$ERR	G$ERR	ERROR AND WARNING ROUTINE
	G$ERRD	ERROR MESSAGE DATA
		SPECIAL LIBRARY ROUTINES
CONTR	CONTR	CONTRACT NODE SET
EXPAND	EXPAND	EXPAND CONTARCTED NODE SET
NODSET	NODSET	OBTAIN SET OF CONTRACTED NODES

FUNCTION TABLE.

NAME	NO. ARGS	ARGS.	MEANING	FCT TYPE
			--- LIST FUNCTIONS: ---	
FIRST DFIRST LAST DLAST	1	LIST	RETURNS THE FIRST OR LAST ELEMENT OF THE LIST, WITH 'D', IT ALSO DELETES THE ELEMENT FROM THE LIST.	TYPE OF LIST
			--- SET FUNCTIONS: ---	
CREATE	1	ZERO	CREATED ELEMENT AS AT.SET	SET
CREATE	2N	PROP.NAME AND VALUE IN PAIRS	GIVES AT.SET WITH MATCHING PROPERTIES, CREATES ONE IF NONEXISTENT.	SET
ATOM	1	INT.	AT.SET WITH THE GIVEN SEQ.NUMBER	SET
ELT	2	INT,SET	AT.SET IN SPEC. PLACE IN THE SET	SET
INDEX	2	AT.SET,	INDEX NO. OF AT.SET IN THE SET.	INT
SIZE	1	SET	NUMBER OF ELEMENTS IN THE SET	INT
PARITY	1	SET	TRUE FOR ODD, FALSE FOR EVEN NUMBER OF ELEMENTS.	LOG
COUNT	1	ZERO	MAXIMAL SEQUENCE NUMBER	INT
COUNT	1	AT.SET	SEQUENCE NUMBER OF THE AT.SET	INT
SUBSET	3	SET,LOG	ELEMENTS OF FIRST,DUMMY ARGUMENT WHICH SATISFY LOG.EXPR.	SET
CHECK	2	PROP.NAME AT,SET	TRUE IF PROPERTY IS DEFINED FOR AT,SET, OTHERWISE FALSE	LOG
			--- PROPERTY FUNCTIONS: ---	
'PROP NAME'	1	AT.SET	RETURNS THE ASSIGNED PROP.VALUE	TYPE OF PROP
			--- GRAPH FUNCTIONS: ---	
NODES	1	GRAPH	NODES OF THE GRAPH	SET
ARCS	1	GRAPH	ARCS OF THE GRAPH	SET
INC	2	GRAPH, SET	UNION OF ALL BOUNDARY NODES OF THE GIVEN SET OF ARCS.	SET
STAR	2	GRAPH, SET	UNION OF ALL ARCS INCIDENT TO THE GIVEN SET OF NODES.	SET
BD	2	GRAPH, SET	SYMMETRIC SUM OF ALL BOUNDARY NODES OF ARCS IN THE SET.	SET
COB	2	GRAPH, SET	SYMMETRIC SUM OF ALL ARCS INCIDENT TO THE NODES IN THE SET.	SET
ADJ	2	GRAPH, SET	UNION OF ALL NODES ADJACENT TO THE NODES IN THE SET.	SET

STATEMENTS

CATEG.	FORM	MEANING
DECLARATION		
	SET X,Y,...	X,Y,... ARE SET VARIABLES.
	'TYPE' STAQUE L,P,...	L,P,... ARE LISTS OF 'TYPE'= REAL, INTEGER, LOGICAL OR SET.
	'TYPE' PROPERTY A,B,...	A,B,... PROP.FUNCTIONS OF 'TYPE' REAL, INT., LOG. OR SET.
	GRAPH G('MOD'),T('MOD'),...	G,T,... ARE GRAPH OF 'MOD'= DIRECTED OR UNDIRECTED, AND PSEUDO-,MULTI- OR NODE-GRAPHS.
ASSIGNMENT		
	S='SET-EXPRESSION'	SET ASSIGNMENT
	L=...:X:L:Y:...	LIST ASG.: X,L,Y,... ARE CONST.,VBLE. OR LIST OF SAME TYPE.
	P(X)='EXPRESSION'	PROPERTY 'P' ASSIGNED TO ATOMIC SET X WITH VALUE 'EXPRESSION'
GRAPH		
	ASSIGN G,X	AT.SET X ASSIGNED TO G AS NODE.
	ASSIGN G,X-Y	ATOMIC SETS X,Y ARE ASSIGNED AS ADJACENT NODES IN GRAPH G.
	ASSIGN G,X-Y,Z	AT.SETS X,Y,Z ASSIGNED TO GRAPH G AS ARC Z WITH END-NODES X,Y.
	DETACH G,S	ELEMENTS OF SET S ARE REMOVED FROM GRAPH G.
ITERATIVE		
	DO 'ST#' FOR ALL X,IN,S	EXECUTES STATEMENTS THROUGH ONE LABELED WITH 'ST#' FOR EACH ELEMENT X OF SET S
	DO 'ST#' WHILE LOG.EXPR.	WHILE LOG.EXPRESSION IS TRUE
REMOVE		
	REMOVE S,T,...,A,B,...	REMOVES SETS S,T FROM UNIVERSE, PROPERTIES A,B FROM AT.SETS WHERE DEFINED.
SAVE-RESET		
	SAVE G,...,P,...	GRAPHS G,...; PROPERTIES P,... SAVED ON AUXILIARY STORAGE
	RESET G,...,P,...	RESET FROM AUXILIARY STORAGE.